Anwendung des Euler Integrals auf Zählprobleme

Bachelorarbeit

Westfälische Wilhelms-Universität

Fachbereich Mathematik und Informatik

Institut für Analysis und Numerik

Arbeitsgruppe Mathematische Optimierung

vorgelegt von:

Jan Marc Heinrich

448 288

Erstgutachter: Prof. Dr. Benedikt Wirth

Zweitgutachter: Prof. Dr. Jörg Schürmann

Münster, 17.September.2020

0. Abstract

Die folgende Arbeit soll die Grundlagen der Zielaufzählung bezüglich der Euler Charakteristik aufgreifen und einen Weg zur nummerischen Implementierung finden, in dem die Ideen aus den Arbeiten von Yury Baryshnikov und Robert Ghrist wie *„Target Enumeration via Euler Characteristic Integrals“* (Yury Baryshnikov & Ghrist, 2009) umgesetzt werden. Dabei soll eine nummerische Betrachtung der Verhaltensweise des resultierenden Integralbegriffes in der zweidimensionalen Ebene entstehen, realisiert durch eine nummerische Testumgebung. Daraus sollen Erkenntnisse über das Approximative Verhalten des Integrales bezüglich der Eulercharakteristik auf diskreten Netzwerken, wie auch über das generelle Verhalten dieses Integralbegriffes gewonnen werden. Zudem werden Werkzeuge entwickelt, die die Approximationsbetrachtung vereinfachen, oder Informationen über das Verhältnis zur kontinuierlichen Lösung liefern sollen.

Inhaltsverzeichnis

1. Einleitung

1.1 Sensoren und Sensornetzwerke

Sensoren sind sowohl in der Industrie als auch im Alltag stark vertreten. Etliche in jedem Smartphone, das ein jeder mit sich führt. Auf den Verkehrswegen, beispielsweise in intelligenten Ampelschaltungen und der Stauerkennung sowie im Auto selber, oder in automatisierten Fertigungsanlagen von Waren bis hin zu Robotersteuerung. Es gibt etliche verschiedene Arten von Sensoren, wie beispielsweise Farb- Temperatur- oder Infrarotsensoren, deren Wirkungsweise, Wirkungsbereich, Größe und deren Berechnungsaufwand beim Auslesen stark unterschiedlich ausfallen können.

Um die steigende Aktualität der Sensorik und ihren Problemklassen angemessen hervorzuheben, sollen vorerst die Aspekte der Sensoren wie: „Entwicklung der Größe von Sensoren, des Preises und der Effizienz“ und „Entwicklung der Vernetzung (von Sensoren)“ betrachtet werden. Problematiken, die Sensoren im Allgemeinen oft haben, wie die Zielidentifikation, führen dann zu anforderungsarmen Voraussetzungen der Sensoren für die Zielaufzählung.

1.1.1 Entwicklung

Die stetige Weiterentwicklung der CPU (Central Processing Unit) , des Arbeitsspeichers und der Vernetzungstechnologie treibt, ganz abgesehen von der Verbesserung der Sensoren selbst, auch die Weiterentwicklung und die Finanzierbarkeit von Sensoren und Sensornetzwerken voran. Das wahrscheinlich beste Beispiel ist hier wieder das Smartphone. Heutzutage gehören Sensoren wie GPS, Temperatursensor, Gyrosensor, Kamera, Mikrophon oder Pulssensor inclusive Vernetzungstechnologie zu den Standards. Der Preis und die Baugröße vieler Sensoren haben durch Weiterentwicklung und maschinelle Produktion stark abgenommen, die Effizienz zugenommen.

Viele Sensoren brauchen zudem Berechnungen zum Auswerten der Sensordaten oder die Daten berechnen sich aus der Zusammenfassung vieler Sensoren oder Sensordaten zu einer Sensorauswertung, wie es beispielsweise das Navigationssystem macht. Mehrere Satellitensignale werden ausgewertet und miteinander verrechnet, um die aktuelle Position zu erhalten. Auch optische Sensoren benötigen oft eine Auswertungssoftware, also höhere Rechenleistungsanforderungen, die möglicherweise in der Nähe des Sensor zur Verfügung gestellt werden müssen. Soll ein optischer Sensor ein Gesicht erkennen, so benötigt es komplizierte Auswertungsalgorithmen und möglicherweise den Abgleich mit einer Datenbank,

auf der Erkennungsmerkmale hinterlegt sind. Im Fall eines Sensornetzwerkes muss möglicherweise nicht jeder Sensor eine eigene Recheneinheit besitzen, sondern es können zentrale Recheneinheiten über Kommunikationswege genutzt werden. Dann müssen teilweise große Datenmengen, wie in dem Beispiel optischer Sensoren auch unbearbeitetes Bildmaterial übermittelt werden, es wird also auch eine hohe Datenübertragungsmöglichkeit benötigt. Die Daten müssen entweder direkt vom Sensor lokal bearbeitet werden, oder eben an eine Zentrale Recheneinheit übermittelt werden. Die Rechenleistungsanforderungen der Sensorauswertung und die Übertragungsrate von Daten sind somit auch entscheidende Faktoren eines Sensornetzwerkes.

Ein weiterer Faktor ist zudem auch die Entwicklung der Datenkommunikationswege. Durch Vernetzungstechnologien wie (Mobil-) Funk, Bluetooth und Glasfaser ist Sensorkommunikation nahezu in Echtzeit möglich, teilweise mit riesigen Datenmengen. Natürlich benötigen Sensoren dann eine lokale Schnittstelle, die den entsprechenden Kommunikationsweg zur Verfügung stellt. Wie oben angemerkt werden Sensoren immer kleiner und dennoch präziser, und durch fortschreitende Kommunikationstechnik wird auch die Übertragungsrate vernetzter Sensoren immer größer, sogar im drahtlosen Bereich, und lokale Kommunikationsschnittstellen werden auch preislich immer erschwinglicher, mit teilweise nur noch millimetergroßen Gehäusen. Netzwerke von Sensoren können also mit immer mehr Sensoren, da ja die Sensoren kleiner und deren Vernetzung günstiger wird, ausgestattet werden und immer breiter und detaillierter Gebiete abtasten, da schnelle und vor allem drahtlose Vernetzung realisierbar ist.

Als weiteren Punkt gilt es anzumerken, dass Sensoren schon in vielen Bereichen der Anwendung vorhanden sind, und somit bestehende Sensoren und Sensornetzwerke auch genutzt werden können, um neue Probleme zu lösen, ohne neue Sensoren hinzufügen zu müssen, oder zumindest nur mit wenigen neue Sensoren auch das neue Ziel zu erreichen. Möchte man die Bewegung der Menschen im Alltag nachvollziehen, so könnte das, abgesehen von geltenden Datenschutzbestimmungen, anhand der Mobilfunkdaten gemacht werden. Möchte man die ungefähre Anzahl täglich bewegter Fahrzeuge in einem Land berechnen, so können die Daten der Verkehrserfassung genutzt werden. Oder die Anzahl von Passanten innerhalb einer Stadt könnte anhand der vorhandenen Überwachungskameras ausgewertet werden. Wettersensoren, Erschütterungssensoren der Erdbebenerkennung, weitere Sensordaten der Forschungsstationen, Verkehrssensoren oder die unter anderem optischen Sensoren der Satelliten sind nur einige Beispiele von nahezu weltweit flächendeckenden Sensornetzwerken.

1.1.2 Sensornetzwerke und Kommunikation

Oft reicht ein einzelner Sensor nicht aus, um eine Problem auf Basis der Sensordaten exakt oder exakt genug berechnen zu können. Netzwerke von Sensoren erfassen oder approximieren dann die Problemlösung. Dabei tritt eine neue Problemklasse auf: Die Sensoren müssen miteinander kommunizieren können, also vernetzt sein. Die lokalen Auswertungen müssen interpretiert und zusammengefasst werden. Betrachtet man ein Sensornetzwerk, in dem die Sensoren Berechnungen zur Auswertung benötigen, so gibt es folgende Möglichkeiten der Problemlösung:

Haben Sensoren in einem Sensornetzwerk möglicherweise hohen Auswertungsaufwand, aber nur kleine Datenmengen an auszuwertendem Sensordaten, so ist es Sinnvoll die Daten über Datenkommunikation von einem Zentralrechner des Sensornetzwerkes berechnen zu lassen. Sind anderseits die auszuwertenden Datenpakete sehr groß, der Berechnungsaufwand aber per Sensor handelbar, so ist unter Umständen die Ausstattung jedes Sensors mit einer lokalen Recheneinheit Sinnvoll. Zwischenlösungen durch Aufteilung der Berechnung, falls dies möglich ist, und Verkleinerung/Packen der zu übermittelnden Auswertungsdaten können ebenfalls den Umständen nach in Betracht gezogen werden. Den Umständen nach, meint hier ein Abwägen zwischen Faktoren wie die Anzahl der auszustattenden Sensoren, die Problemgröße des Berechnens der Auswertung im Verhältnis zur benötigten lokalen Rechenleistung, der Preis der lokalen Recheneinheiten sowie die Datenübertragungsrate der Sensorverbindungen. Haben Sensoren in einem Sensormodel einer Problemklasse jedoch geringe Auswertungsanforderungen, also sowohl möglichst kleine Auswertungsdaten und entsprechend geringe Rechenleistungsanforderungen dieser Auswertungen, so stellt dieses Sensormodell die geringsten Anforderungen an die Sensoren, hat also eine dementsprechend große Problemklasse der durch die Sensoren zu lösenden Probleme.

1.1.3 Eigenschaften

Zunächst macht es aber Sinn, eine allgemeine Eigenschaft des Sensors auszuführen. Sensoren überprüfen meist ein lokalen begrenzten Bereich, in dem sie Ereignisse aufzeichnen können. Dieser Bereich ist je nach Sensortechnologie aber auch von Ziel zu Ziel unterschiedlicher Art. Ein Bewegungsmelder hat beispielsweise einen Bereich, in dem er Bewegungen wahrnimmt. Die Größe dieses Bereiches unterscheidet sich möglicherweise je nach Art des Bewegung verursachenden Ereignisses. Die Bewegung kleinerer Objekte werden vielleicht auf geringere Distanz erkannt, wie die Bewegung eines sehr großen Objektes. Die Stärke der Bewegung beeinflusst wohlmöglich auch den Erkennungsbereich, und es gibt sicherlich eine Grenze, ab

der die Bewegung zu schwach ist, um registriert zu werden, oder das Objekt zu klein ist, um erfasst zu werden.

Diese Eigenschaft, also die unterschiedliche Form und Größe des Bereiches, in dem ein Ziel jeweils erkannt wird, haben viele Sensoren, wie Ton-, Licht-, Kamera-, Magnetfeld- oder seismischer Sensor. Dieser Fakt ist besonders in Sensornetzwerken interessant, wenn es um Zielaufzählung geht. Erfassen 2 Sensoren ein Ziel, das im Erfassungsbereich beider Sensoren liegt, so fasst das Sensornetzwerk ein Ziel möglicherweise als mehr als ein Ziel auf. Da sich der Erfassungsbereich von Ziel zu Ziel unterscheiden kann, macht es Sinn den Bereich, in dem ein Ziel erkannt wird, also den Zielträger, dem Ziel zuzuordnen. Jedes Ziel hat somit einen Zielträger, also einen Bereich, in dem es von den Sensoren des Sensornetzwerkes erfasst wird.

1.2 Zielaufzählung

1.2.1 Verschiedene Sensoranforderungen

Bemerkenswert ist überleitend, im oberen Beispiel des Bewegungsmelders werden alle erkannten Bewegungen nur als solche erkannt. Der Sensor liefert keinerlei Information über Größe und Art oder Position, weder der Bewegung noch des verursachenden Objektes. Ob zwei unterschiedliche Objekte die Bewegung verursacht haben, oder dasselbe Objekt, ob es unterschiedliche Bewegungen waren, oder ähnliche, kann nicht geschlossen werden. Diese Eigenschaft hat nicht nur der einfache Bewegungsmelder. Viele Sensoren können Ziele oder Ereignis verursachende Objekte nicht, oder nur unter hohem Rechenaufwand unterscheiden. Soll anhand einer Kamera beispielsweise die Anzahl unterschiedlicher durchlaufender Passanten innerhalb eines Stadtbereiches registriert werden, so muss ein Algorithmus an Merkmalen wie denen des Gesichtes, oder der Körpergröße und anderen die Zielidentifikation hinterlegen, speichern und vergleichen. Eine alternative Lösung dieses Problems ist das Aufstellen mehrerer Kameras, die einfach nur die Anzahl der Bewegungen registrieren. Gibt es genügend Kameras, die diese einfachere Aufzeichnung an geeigneten Stellen durchführen, so kann nach den Ideen von Baryshnikov und Ghrist auf die Gesamtzahl der Passanten geschlossen werden (Yury Baryshnikov & Ghrist, 2009). Die Zielidentifikation und die daraus resultierende aufwendige Berechnung wurde vermieden, allerdings werden so möglicherweise mehr Kameras benötigt, wenn sie auch über keine große lokalen Rechenleistungen verfügen müssen . Die Verwendung einfacher Sensoren wie Bewegungsmelder machen das Problem dann auch Wirtschaftlich noch attraktiver, da die lokale Auswertung entfällt und der einzelne Sensor sicherlich günstiger ausfällt. Die Allgemeinheit der Problemklasse, nur die Zählfähigkeit und insbesondere keine Zielidentifikation zu fordern und zudem den Zielen zugehörige Zielträger

zuzuordnen, ergibt in diesem Fall eine alternative Berechnungsmöglichkeit der Problemstellung, ganz abgesehen von einer Breite der Anwendung im Allgemeinen. Viele Zählprobleme können somit auf die auf folgende Problemklasse übertragen werden.

1.2.2 *Target enumeration via euler characteristic integrals'*

Der folgende Abschnitt ist ein inhaltlicher Überblick der Arbeiten von Yury Baryshnikov und Robert Ghrist *„Target Enumeration via Euler Characteristic Integrals“* (Yury Baryshnikov & Ghrist, 2009), welche die Problemklasse der Zählung der Gesamtzahl beobachtbarer Ziele in einer Region mit Hilfe lokaler Zählungen löst, die von einem Netzwerk von Sensoren durchgeführt werden, von denen jeder die Anzahl der Ziele in der Nähe misst. Die folgende Arbeit basiert auf diesen Erkenntnissen, weshalb die relevanten Erkenntnisse hier noch einmal aufgeführt werden. Der Hinführung halber wird vorerst ein Feld unendlich kleiner Sensoren in jedem Punkt eines glatten Bereiches betrachtet und angenommen, dass eine endliche Anzahl an Zielen vorliegt. Ein Sensor bei $x \in \mathbb{R}^2$ liefert eine quantisierte Zählung $h(x) \in \mathbb{N}$ entsprechend der Anzahl der Ziele „in der Nähe“. Die Sensoren sind dabei dicht verteilt und haben keine Information über Standort und Zielidentität der Ziele. In der Folgenden Abbildung 1 sind drei Ziele Oi mit ihren Zielträgern Ui für i =1,2,3 im Sensorraum W zu sehen. Zählsensoren im weißen Bereich geben den Wert 0 zurück, im hellblauen Bereich den Wert 1.

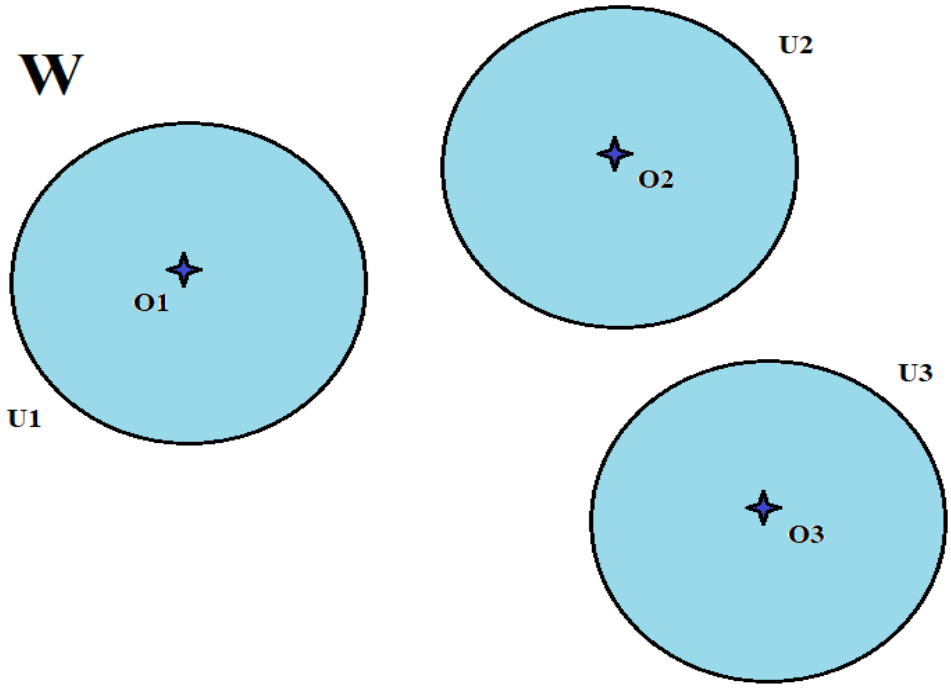

Abbildung 1: Kontinuierlicher Zielraum W mit Zielen Oi und Zielträgern Ui in W.

Unter der kritische Annahme, jedes Ziel Oα beeinflusst seine Umgebung Uα in einem Abstand R >0 um Oα , d.h. wird in diesem Bereich von den Sensoren erfasst, und der Voraussetzung, dass alle Ziele auf homogener Domäne identisch sind, so folgt aus der Linearität des Integrales

$$\#\{O\alpha\} = \frac{1}{M}\int h(x)\,dx,$$

wobei M = πR^2 den Flächeninhalt der Umgebung Uα auf der das Ziel erkannt wird bezeichnet.

In der Abbildung 1 kann man also einfach das herkömmliche Integral der Höhenfunktion h berechnen, und durch das Maß der Zielträger teilen, um die Gesamtzahl der Zielträger zu erhalten. Dies ist sofort auf beliebige Dimensionen übertragbar und auch auf allgemeinere *Zielauflagen* Uα, solange die Uα für jedes α die gleiche $Masse$ haben. Zudem ist es auch möglich den Wertebereich zu diskretisieren, also z.B. h(x) auf einem Gitter, einer endlichen Menge abzutasten. Ersetzt man nun geometrische durch topologische Annahmen, also durch variable *Zielhalter* $U\alpha$ fester Topologie, lässt diese Methode sich auf ein breites Spektrum von Aufzählproblemen in Sensornetzwerken verallgemeinern, denn die Voraussetzung der Zielträger gleicher Masse wird durch die gleicher Topologie ersetzt. Dafür muss, wie in der herkömmlichen Integrationstheorie für das Maß ein Weg gefunden werden, die Topologie eines Zielträgers mathematisch zu erfassen, also einem Wert zuzuordnen. Dies ist die Euler Charakteristik, eine topologische Invariante, die im Folgenden Abschnitt genauer betrachtet wird. Unter der Annahme, dass die Zielträger gleicher Topologie sind, also auch gleicher Euler Charakteristik, so kann man die Anzahl der Ziele dann durch Integration mittels der Euler Charakteristik berechnen. Für gegebene *Zählfunktion* h: X → ℕ für $\{U\alpha\}$ eine Sammlung kompakter Zielträger in einem Sensorraum X, die $\chi(U\alpha) = N \neq 0 \ \forall\, \alpha$ für N ∈ ℕ erfüllen, also gleiche Eulercharakteristik besitzen, kann die Anzahl der Ziele folgend berechnet werden:

$$\#\alpha = \frac{1}{N}\int_{\boldsymbol{X}} h\,dx$$

$$Beweis\colon \quad \int_{\boldsymbol{X}} h\,d\chi = \int_{\boldsymbol{X}} \left(\sum_{\alpha} \mathbf{1}_{U\alpha}\right) d\chi = \sum_{\alpha}\int_{\boldsymbol{X}} \mathbf{1}_{U\alpha}\,d\chi = \sum_{\alpha}\chi(U\alpha) = N\,\#\alpha \ \blacksquare$$

Für den Wert N = 0 ist keine Berechnung möglich. Im Fall N = 1 (konvexe Zielsätze) ist die Berechnung sehr einfach.

Wie in der einleitenden Problemstellung erwähnt beruhen die mathematischen Werkzeuge auf Sensordaten in jedem Punkt eines kontinuierlichen Sensornetzwerkes. Konkrete Anwendungen liefern aber äußerst selten ein kontinuierliches Sensornetzwerk, weshalb das Verhalten auf

diskreten Sensornetzwerken untersucht werden muss. Folgendes Korollar aus *„Target Enumeration via Euler Characteristic Integrals“* liefert für dieses Problem bereits eine Antwort:

Baryshnikov und Ghrist: ***Korollar 5.2:***

Nimmt man eine obere semikontinuierliche konstruierbare Funktion $h\colon \mathbb{R}^2 \to \mathbb{N}$ *an, und sei G ein Netzwerkgraph mit Knoten* $N \subset \mathbb{R}^2$*, wobei nur die Beschränkung von h auf N bekannt ist (insbesondere sind die Koordinaten von* $N \subset \mathbb{R}^2$ *unbekannt). Wenn das Netzwerk G die Konnektivität der oberen und unteren Auslenkungssätze von h korrekt abtastet, dann gibt die spezialisierte Berechnungsformel die genaue Anzahl der Ziele zurück.*

Die Gruppe der konstruierbaren Funktionen meint Funktionen $h\colon X \to \mathbb{Z}$ mit endlichem Definitionsbereich und definierbaren Trägern $h^{-1}(c)$.

Dieses Korollar gibt zusammen mit den im nächsten Abschnitt aufgeführten Eigenschaften des Integralbegriffes wie dem *Fubini Theorem(2.8)* (Yury Baryshnikov & Ghrist, 2009) nun eine geeignete Grundlage für die Anwendung auf diskrete Sensornetzwerke. Bei korrekter Abtastung der Zielträger kann das Integral anhand endlich vieler gut platzierter Sensoren exakt wiedergegeben werden, und dass ohne Wissen über die Positionen der Sensoren. Im weiteren Verlauf der Arbeit werden die Auswirkungen von Ungenauigkeiten der Abtastung durch diskrete Sensornetzwerke getestet und mit Arbeiten zu diesem Aspekt verglichen. Verhält sich die Approximation des Integrales mittels Euler Charakteristik ähnlich zu dem herkömmlichen Integralbegriff?

Eine weitere Eigenschaft bezüglich der numerischen Approximation ist die Möglichkeit, lokale Berechnungen zu einer globalen Lösung zusammenzusetzen, also das Gebiet in Bereiche zu unterteilen und die Problemlösung durch lokale Lösungen zusammenzusetzen. Dies gibt in einem Sensornetzwerk die Möglichkeit, Teilberechnungen lokal durchzuführen und zu einem Gesamtergebnis zusammenzufügen, und so den Rechenaufwand aufzuteilen, oder Vorhandende lokale Rechenmöglichkeiten zu integrieren.

Die Eigenschaften des Integralbegriffes bezüglich der Euler Charakteristik zusammen mit der Möglichkeit der Diskreditierung genügen nun, um verschiedene Zielaufzählproblemklassen zu lösen. Baryshnikov und Ghrist lösen vier Problemklassen, darunter die Problemklasse Beweglich Ziele, also Ziele, welche sich entlang kontinuierlichen Pfaden innerhalb eines festes Zeitintervalls bewegen. Unter der Annahme, jeder Sensor kann ein Ziel erkennen, dass in die

Nähe des Sensors kommt und dann einen eigenen Zähler erhöht, kann somit die Gesamtzahl der Ziele berechnet werden.

1.3 Integration mittels Euler Charakteristik

1.3.1 Euler Charakteristik

Sollen geometrische durch topologische Annahmen der Zielträger ersetzten werden, so muss eine geeignete Theorie bereitgestellt werden, um die Topologie der Zielträger genauer zu Charakterisieren. Wie in der herkömmlichen Integrationstheorie Mengen ein Maß zugeordnet wird, so sollen auch den Objekten eine topologische Invariante zugeordnet werden, eben der Euler Charakteristik. Objekte werden in diesem Sinn meist einem Kettenkomplex zugeordnet.

Ein Kettenkomplex (C,d) ist definiert als eine Folge $C_n, n \in \mathbb{Z}$ von R-Modul zusammen mit einer Folge $d_n: C_n \to C_{n-1}$ von R-Modul-Homomorphismen mit der Eigenschaft, dass

$d_n \circ d_{n+1} = 0$ für alle n gilt. Man bezeichnet dabei d_n als *Randoperator* und die Elemente von C_n als *n-Ketten*. Ein *n-Zykel* ist ein Element von $Z_n(C,d) = kern\ d_n \in C_n$, ein *n-Rand* ist ein Element von $B_n(C,d) = bild\ d_{n+1} \in C_n$. Durch die Bedingung $d_n \circ d_{n+1} = 0$ ist jeder *Rand* ein *Zykel*, und somit ist der Quotient definiert als

$H_n(C,d) = Z_n(C,d) \,/\, B_n(C,d)$, der *n-ten Homologiegruppe* von (C,d).

Für (A_n, d_n) einen Kettenkomplex, bei dem alle bis auf endlich viele A_n null sind und die übrigen endlich erzeugt sind, kann die Euler-Charakteristik durch:

$$\chi = \sum_n (-1)^n\ rang(A_n)$$

definiert werden.

Zudem kann man die Euler-Charakteristik auch über die Homologiegruppen definieren:

$$\chi = \sum_n (-1)^n\ rang(H_n)$$

Diese Eigenschaft macht es möglich, die Euler Charakteristik eines Komplexes anhand seiner „Löcher" zu definieren. Diese alternative Definition wird beispielsweise für den Beweis des Theorems 4.3 (Yury Baryshnikov & Ghrist, 2009) genutzt. Die unterschiedlichen Definitionen der Euler Charakteristik schaffen also die Möglichkeit die Stärken der jeweiligen Definition auszuschöpfen.

Es gibt verschiedene Ansätze der Realisierung des Kettenkomplexes. Eine geometrische Lösung im $\mathbb{R}^n$ stellt der Simpliziale Komplex dar, über diesen dann die Simpliziale Homologie definiert werden kann. Dabei wird ein Objekt im $\mathbb{R}^n$ einem Kettenkomplex zugeordnet, die Ketten sind m-dimensionale (euklidische) Simplexe im $\mathbb{R}^n$ und der Randoperator ist der Operator zu jedem Simplex, der auf den geometrischen Rand des Simplizes abbildet, also die Zusammensetzung der Simplizes, wie zum Beispiel in Abbildung 2 dargestellt ist. Das dunkelblaue Objekt wird durch Simplizes dargestellt. Dabei bilden nicht nur die Dreiecke Simplizes, sondern auch die Seitenflächen. Die Art der Zusammensetzung dieser Simplizes bildet dann den Randoperator.

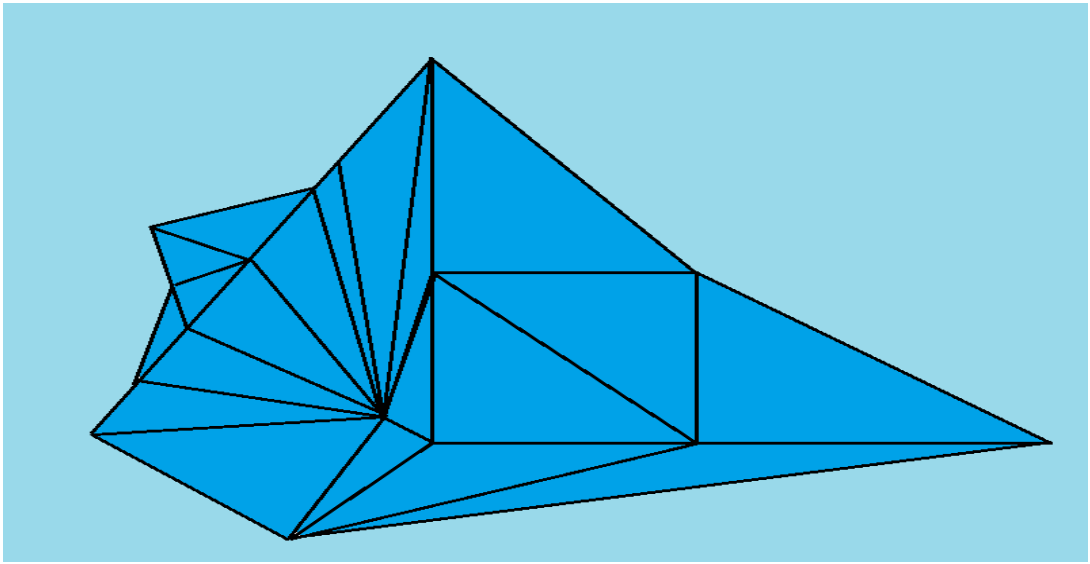

Abbildung 2: Simpliziale Triangulierung eines zweidimensionalem Objektes

Da nicht jedes Objekt im $\mathbb{R}^n$ eine Darstellung durch einen Simplizialen Komplex besitzt, und anderseits die Darstellung als Simplizialer Komplex keine eindeutige ist, da es verschiedene unterschiedliche Möglichkeiten gibt einem Objekt einen geeigneten Simplizialen Komplex zuzuordnen, und somit also keine Topologische Invariante vorliegt, ersetzt man die m dimensionalen Simplizes durch stetige Abbildungen der m-Simpizes in den Komplex. Dies ist die singuläre Homologiegruppe. Einen anderen Ansatz bildet der CW-Komplex, dessen Elemente homöomorph zu $[0,1]^k, k = 0, \dots, n$ für einen n-dimensionalen *CW-Komplex* sind, oder auch die Homotopiegruppe, deren Elemente durch stetige Abbildungen der n-dimensionalen Einheitssphäre in den Raum X gegeben sind.

Mittels singuläre Homologie kann man die sogenannten *Betti-Zahlen* (Betti, 1870) definieren, positive ganzzahlige Zahlen, die die globalen Eigenschaften eines topologischen Raumes beschreiben. Diese lassen sich auch über die singulären Homologiegruppen definieren und dies ergibt insbesondere einen Zusammenhang mit der *Euler-Charakteristik*. Dieser Zusammenhang

wird in der nummerischen Realisierung der Berechnung der Zielgesamtzahl bei Zielaufzählungsproblemen in planaren Netzwerken genutzt. Dazu ist auch die geometrische Interpretation der nullten Betti Zahl interessant.

1.3.2 Betti Zahlen - Zusammenhangskomponenten

Für einen topologischen Raum X ist die *i-te Betti-Zahl* von X als $b_i(X) = \dim_{\mathbb{Q}} H_i(X, \mathbb{Q})$ mit $i \in \mathbb{N}_0$ definiert. $H_i(X, \mathbb{Q})$ bezeichnet dabei die *i-te* singuläre Homologiegruppe mit Koeffizienten in den rationalen Zahlen.

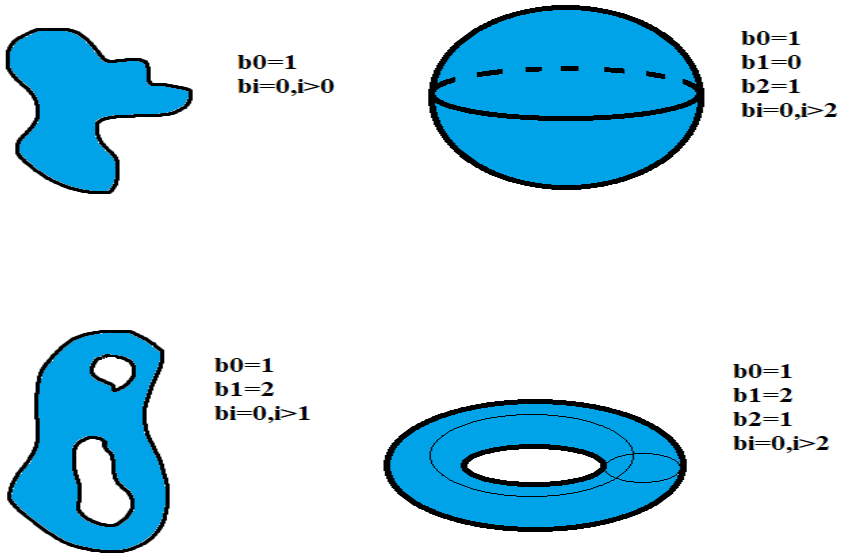

Abbildung 3: Die Betti Zahlen von vier verschiedenen Objekten

Es beschreibt b_0 ist die Anzahl der *Wegzusammenhangskomponenten*, b_1 ist die Anzahl der *„Zweidimensionalen Löcher“* und b_2 ist die Anzahl *dreidimensionalen Hohlräume* eines topologischen Raumes X. Es gilt außerdem $b_m = 0$ für alle $m > n$ für eine n-dimensionale Mannigfaltigkeit. Die Abbildung 3 zeigt mehrere Objekte und ihre Betti Zahlen. Außerdem existiert ein Zusammenhang zur Euler Charakteristik. Die Betti Zahlen stellen zudem Topologische Invarianten dar. Außerdem ist die Euler-Charakteristik ist die alternierende Summe der Betti-Zahlen:

$$\chi(X) = \sum_{i \in \mathbb{N}_0} (-1)^i \, b_i(X)$$

Die Nullte Betti Zahl, also die Anzahl der Wegzusammenhangskomponenten. Zumindest im zweidimensionalen kann optisch sehr leicht erkannt werden, wie viele Zusammenhangskomponenten ein Objekt hat. Auch nummerisch lässt sich der Begriff Zusammenhangskomponente in einem Graphen mit ungerichteten Verbindungen leicht charakterisieren. Dies bildet die Grundlage der hier Verfolgten nummerischen Umsetzung der

Zielaufzählung und folgt den Ideen von Baryshnikov und Ghrist in Sektion 5.3 von *„Target Enumeration via Euler Characteristic Integrals“*. In eben dieser Veröffentlichung wird auch die Integration mittels der Euler Charakteristik eingeführt.

1.3.3 Integration

Ähnlich zur herkömmlichen Integrationstheorie können gewisse Mengen gebildet werden, die keine Wohldefinierte Eulercharakteristik besitzen. Deshalb wird die Anwendung auf ein o-minimal System beschränkt, ein Mengensystem, in dem dann auf Grund des Triangulationstheorems (Kneser, 1926) jede Menge eine wohldefinierte Euler Charakteristik besitzt (Yury Baryshnikov & Ghrist, 2009). Vereinfacht ist ein o-minimal System A eine Folge boolescher Algebren A_n von Teilmengen des $\mathbb{R}^n$, eine Familien von Mengen, die mit den Operationen Schnittmenge und Komplement abgeschlossen sind. Das System A ist dabei abgeschlossen unter Produkten und Projektionen. Elemente von $A_n$werden als definierbar bezeichnet, und eine Funktion heißt definierbar, falls ihr Graph einer definierbaren Menge entspricht.

Des Weiteren ist die Euler Charakteristik eine ganzzahlige Topologische Invariante, somit wird eine Integration in die Ganzen Zahlen erreicht, nicht etwa in die reellen Zahlen. Für die Approximation ist dieser Aspekt von besonderer Interesse, da eine Annäherung nur in ganzzahligen Schritten geschehen kann. Zu betrachten sind also CF(X) die Gruppe der konstruierbaren Funktionen eines topologischen Raumes X, also Funktionen $h: X \to \mathbb{Z}$ mit endlichem Definitionsbereich und definierbaren Trägern $h^{-1}(c)$.

Nach dem Triangulationstheorem (Kneser, 1926) kann also man jedes Element $f \in CF(X)$ als Summe $f = \sum_\alpha c_\alpha \mathbf{1}_{\sigma_\alpha}$ für $c_\alpha \in \mathbb{Z}$ und $\sigma_\alpha \subset X$ einem Simplex schreiben. Das Euler Integral ist dann der Homomorphismus $\int_X d\chi: CF(X) \to \mathbb{Z}$ der $\sum_\alpha c_\alpha \mathbf{1}_{\sigma_\alpha} \to \sum_\alpha c_\alpha \chi(\sigma_\alpha)$.

Das Integral kann noch auf andern Arten interpretiert werden und die Funktionenbasis erweitert werden. Ein nährreicher Boden wird durch den moderneren Gabenansatz geliefert, indem das Integral mittels Euler Charakteristik als Direktes Bild (*Push Forward*) interpretiert wird, und somit als Funktional interpretiert werden kann. Daraus ergeben sich starke Werkzeuge wie beispielsweise das *Fubini Theorem* (Yury Baryshnikov & Ghrist, 2009). Da diese Arbeit die angesprochenen Aspekte nicht weiter vertieft, wird an dieser Stelle auf die Veröffentlichung verwiesen.

1.4 Bezug zu weiteren Arbeiten

Die Möglichkeit der Darstellung eines Objektes durch verschiedene Netzwerke und Sensorwerte führt schnell auf die Idee eines Minimalen Netzwerkes, dass noch ausreicht, die Charakteristik genau zu berechnen. Die Theorie dieses Ansatzes wird in (Tanaka, 2017) behandelt. Für den weiteren Verlauf relevant ist vor allem folgende Erkenntnis. Die Minimierung eines diskreten Sensornetzwerkes ändert den Wert des darauf berechneten Euler Integrales nicht. Dieser Sachverhalt soll vor allem für die Optimierung und Anwendbarkeit des Integrales genutzt werden. Diese Idee und eine Herleitung mehr aus der nummerischen Sichtweise wird in Abschnitt 2.3 behandelt. Zudem werden weitere Eigenschaften der Minimierung vorgestellt.

Der ganzzahlige Integralbegriff kann auf reelwertige Funktionen erweitert werden (Yuri Baryshnikov & Ghrist, 2010). In dieser Umsetzung wird auf diesen Aspekt nicht weiter eingegangen, dennoch ist diese Möglichkeit ein erwähnenswertes Detail, da die Approximationsbetrachtung des diskreten ganzzahligen Integralbegriffes möglicherweise dahingehend erweitert werden kann.

Eine ähnliche Betrachtung der nummerischen Umsetzung des ganzzahligen Euler Integralbegriffes auf zweidimensionalen Gebieten wie in dieser Arbeit behandelt wird durch Kruppa betrachtet (Krupa, 2012). Dabei wird das Verhalten des Euler Integrales von endlichen runden Zielträgern auf rechteckiger Ebene betrachtet. Eine wichtige Erkanntnis für die weitere Betrachtung ist folgende: Die Zielträger dürfen nicht den Rand des Gebietes reichen, sonst ergeben sich Fehler. In den Bildern aus Abschnitt 2.5 wurde dies beachtet. Ebenso wird das Problem der Überlappung zweier Kreise in einem sehr kleinen Bereich betrachtet. Stellt man einen Kreis diskret dar, wie in Abschnitt 2.5 beschrieben wird, und schneidet sich dieser mit einem anderen Kreis am Rand, so können auf Grund der endlichen Genauigkeit Bereiche an den Berührungspunkten entstehen, die als Loch dargestellt werden, aber eigentlich nicht existent sind. Genauere Grafiken dazu sind in der genannten Veröffentlichung zu finden. Die hieraus gewonnenen Erkenntnisse sind vor allem für die Bilder 2 und 3 aus der Approximationsanalyse in 2.5.1-3 bedeutend.

2. Anwendung der Zielaufzählung mittels Euler Integration

2.1 *Baryshnikov und Ghrist:* Theorem 4.3

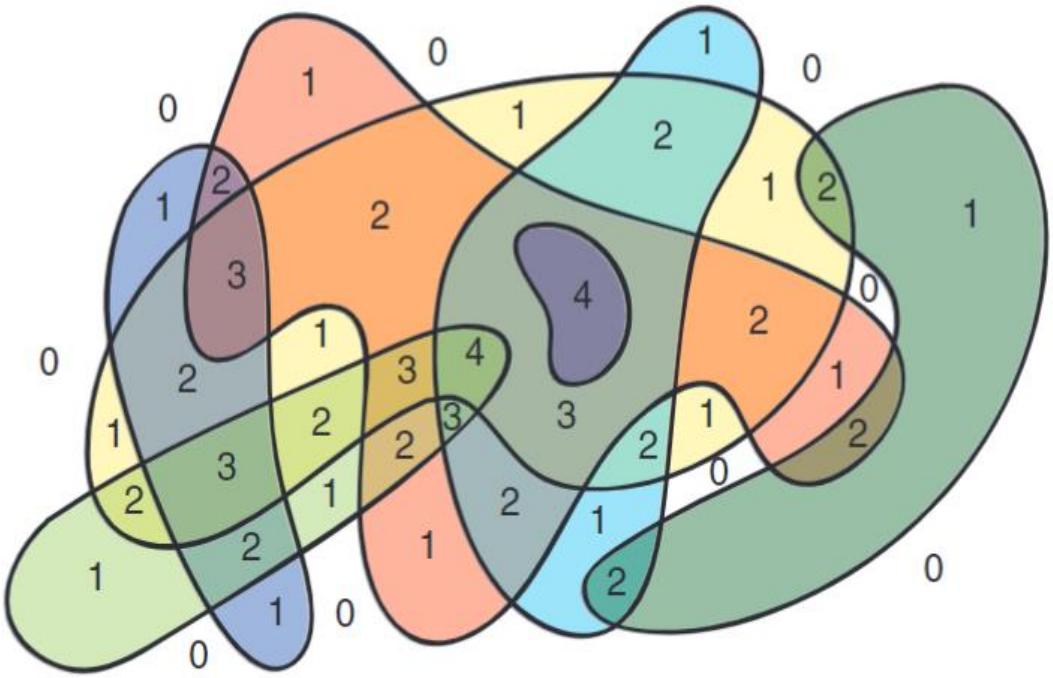

Abbildung 4: Zählfunktion sieben verschiedener Zielträger. (Yury Baryshnikov & Ghrist, 2009)

In Abbildung 4 befinden sich sieben teilweise überlappende Objekte, dessen Anzahl anhand der Daten eines kontinuierlichen Sensornetzwerkes, dessen Sensoren ausschließlich die lokale Anzahl der Objekte wiedergeben können, ermittelt werden soll. Nach der Notation der Höhensätze wie folgt, kann das Integral der Höhenfunktion h, die das Sensornetzwerk zurückgibt, wie folgend berechnet werden. Die Höhensätze $\{h = s\} := \{x \in \boldsymbol{X}: h(x) = s\}$, $\{h > s\} := \{x \in \boldsymbol{X}: h > s\}$, $\{h < s\} := \{x \in \boldsymbol{X}: h < s\}$,… beschreiben die Menge aller Punkte der Funktion h, in denen h jeweils Werte (echt) größer, (echt) kleiner, oder gleich der Ganzen Zahl s annimmt. Nun kann man das Integral der Höhenfunktion nach *Proposition 4.1* aus *„Target Enumeration via Euler Characteristic Integrals“* über die Euler-Charakteristik der Höhensätze berechnen.

Für $h \in CF(\boldsymbol{X})$ *lässt sich das Integral von h im Bezug auf die Euler Charakteristik* $d\chi$ *wie folgt berechnen:*

$$\int_{\boldsymbol{X}} h \, d\chi = \sum_{s=-\infty}^{\infty} s \, \chi(h = s) = \sum_{s=0}^{\infty} \chi(h > s) - \chi(h < -s)$$

Wie in Abbildung 5 veranschaulicht wird, kann nun die Anzahl der Ziele anhand der Höhensätze berechnet werden, und ergibt korrekte Anzahl 7 der Überlappenden Objekte.

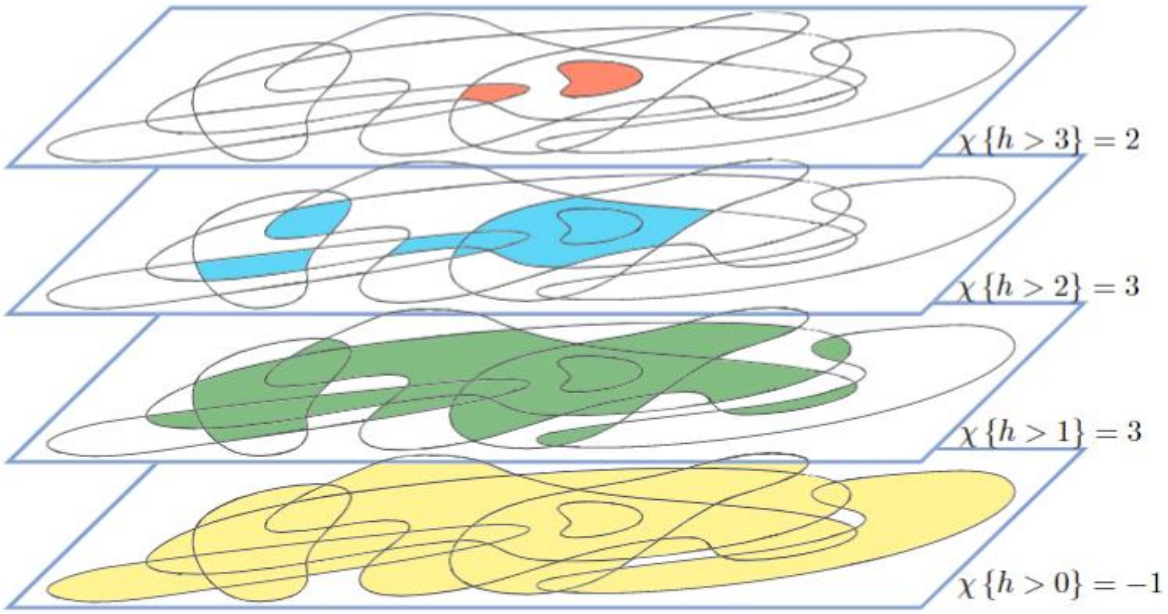

Abbildung 5: Höhensätze der Zählfunktion aus Abbildung 4. (Yury Baryshnikov & Ghrist, 2009)

Zwar wurde nun die Berechnung des Integrals auf Berechnungen der Euler Charakteristik der Höhensätze verlagert, jedoch ist die Berechnung der Euler Charakteristik aus nummerischer Sicht ebenso schwer zu Formalisieren. Ein nummerisch greifbareren Ansatz liefert für zwei dimensionale Sensornetzwerke das *Theorem 4.3* von Baryshnikov und Ghrist, der Integralwert der Höhenfunktion h(x) kann durch die alternierende Summe der nullten Betti Werte der Höhensätze berechnet werden, durch die Anzahl der Zusammenhangskomponenten der jeweiligen Höhensätze:

Theorem 4.3 aus „Target Enumeration via Euler Characteristic Integrals“:

Für eine konstruierbare und obere semi-kontinuierliche Funktion h: $\mathbb{R}^2 \to \mathbb{N}$ *gilt*

$$\int_{\mathbb{R}^2} h \, d\chi = \sum_{s=0}^{\infty} (\beta_0\{h > s\} - \beta_0\{h \leq s\} + 1),$$

wobei β_0 *die Nullte Betti-Zahl bezeichnet. Sie ist gleich der Anzahl der Wegzusammenhangskomponenten von* $\{h > s\}$ *beziehungsweise* $\{h \leq s\}$.

Da die anschließende Analyse der Approximation der Euler Integration auf dem Theorem beruht, wird der Vollständigkeit wegen der Beweis angeführt.

Beweis: *Sei* $A \in \mathbb{R}^2$ *nichtleer und kompakt. Die homologische Definition (Borel-Moore) liefert* $\chi(A) = \sum_{n=0,1,\dots}(-1)^n\, dim(H_n(A))$, *für A kompakt ist* $H_n(A)$ *also die singuläre Homologie. Nach dem Satz in der singulären Homologie (Hatcher, 2002) folgt* $H_s(A) = 0$ *für* $s > 2$, *dass heißt* $\chi(A) = dimH_0(A) - dimH_1(A)$. *Nach der Alexander Dualität*[7] *gilt* $dimH_1(A) = dimH^0(\mathbb{R}^2 - A, A) = dimH_0(\mathbb{R}^2 - A) - 1$ *mit* $dimH_0 = \beta_0$. *Die Behauptung folgt dann durch das Einsetzen der Berechnung mit* $A = \{h > s\}, \mathbb{R}^2 - s = \{h \leq s\}$ *in die Berechnungsformel.* ■

Diese Formulierung ist sehr wichtig in planaren Sensornetzwerken, die Berechnung der Anzahl von Zusammenhangskomponenten der Höhensätze kann gut Formalisiert werden, in Abbildung 5 ist die Anzahl der Zusammenhangskomponenten des Höhesatzes beispielsweise $\beta_0\{h > 3\} = 2$, $\beta_0\{h > 2\} = 3$, $\beta_0\{h > 1\} = 4$ und $\beta_0\{h > 0\} = 1$, wie optisch leicht abzuzählen ist. Der Begriff der Zusammenhangskomponente ist auch nummerisch bereits formalisiert, Graphen mit Knoten und Verbindungen unter den Knoten besitzen Zusammenhangskomponenten, also Knotenmengen, die nur untereinander, und nicht mit knoten außerhalb der Menge verbunden sind.

Wie von Baryshnikov und Ghrist in Sektion 5.3 bereits erwähnt wird, kann so ein Sensornetzwerk als Graph mit Knoten, Knotenwerten und Knotenverbindungen koordinatenfrei dargestellt werden, und das Integral der Zählfunktion über die Anzahl der Zusammenhangskomponenten innerhalb des Graphen berechnet werden. Die Verbindungen der Knoten können dabei beispielsweise der Datenvernetzung des Sensornetzwerkes entsprechen, über welches die Sensoren ihre Informationen austauschen. Zwar lässt sich dieses Verfahren nicht auf höhere Dimensionen erweitern, aber lassen sich einige Probleme höherer Dimension in die zweite Dimension übertragen, um *Theorem 4.3* zur nummerischen Berechnung nutzen zu können. Der folgende Abschnitt beschreibt die nummerische Umsetzung des Theorems und stellt einen Algorithmus in Python vor.

2.2 Programmtechnische Umsetzung über Theorem 4.3

2.2.1 Idee der Vorgehensweise

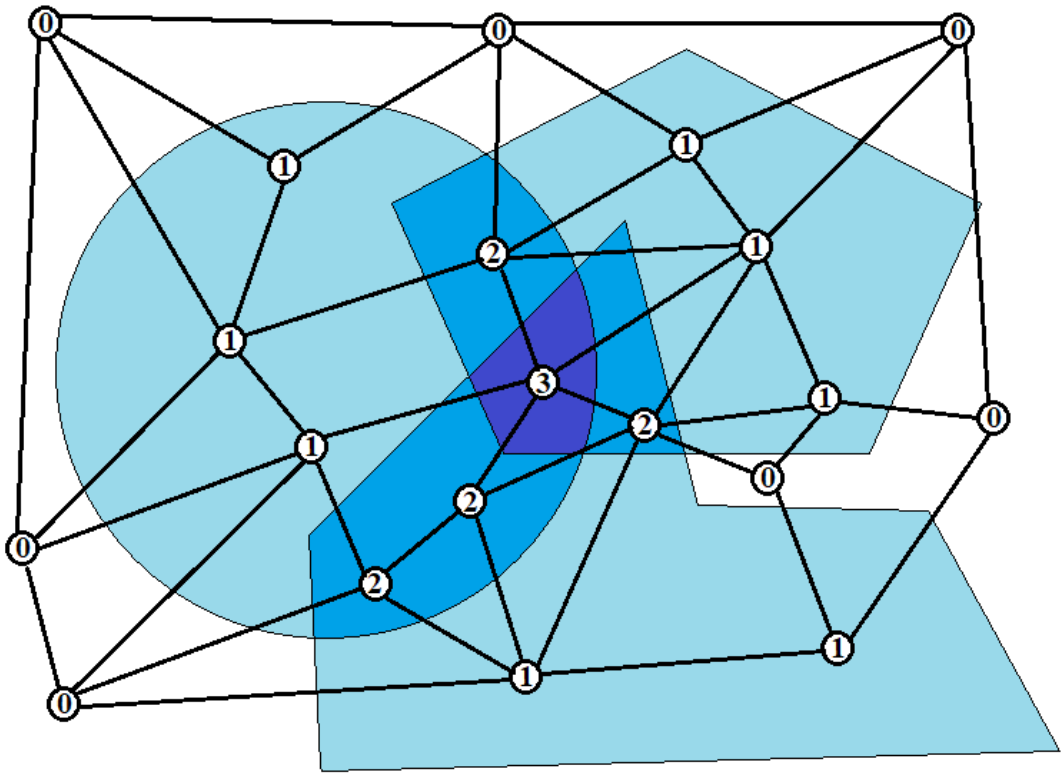

Abbildung 6: Drei Objekte werden durch ein diskretes Sensornetzwerk approximiert. Die Zahlen in den Sensorpunkten entsprechen den Sensorwerten, die Verbindungslinien den Sensorverbindungen.

In der Abbildung 6 sind drei sich teilweise überschneidende Objekte zu sehen, welche von einem diskreten Netzwerk von Sensoren abgetastet werden. Die Sensoren entsprechen den Knotenpunkten mit entsprechendem Sensorwert, die Anzahl der überliegenden Objekte und die Verbindungen der Knoten, wohlmöglich durch die Kommunikation der Sensoren gegeben, den Verbindungslinien zwischen den Knoten. Ein Knoten beziehungsweise Sensor i kennt also einen Knoten j direkt, falls dies entsprechenden Knoten in direkter Verbindung stehen. Jeder Sensor speichert zudem seinen Sensorwert, Knoten j hat somit den Knotenwert, der jeweils für den Knoten angegeben ist.

Soll nummerisch das Integral der Zählfunktion aus *Theorem 4.3* umgesetzt werden, so müssen die Sensoren, also die Knoten, nummeriert werden, um von Knoten i oder j reden zu können. Daraufhin muss zu jedem Knoten ein Knotenwert, der Sensorwert, gespeichert werden. Eine (doppelt verkettete) durchnummerierte Liste, in welcher auf die einzelnen Listenelemente per Index zugegriffen werden kann, eignet sich zur Speicherung der Sensorwerte und zugehöriger

Sensornummer. Das Listenelement fünf enthält dann den Wert des Sensors mit Index 5. Wie in der Informatik üblich beginnt die Nummerierung des ersten Listenelementes mit 0.

Die Verbindungen zwischen den durchnummerierten Sensoren kann in einer Adjazenz Matrix gespeichert werden. Ist in der Zelle (i, j) dieser Matrix der Wert 1 hinterlegt, so kennt Sensor i den Sensor j. Da keine gerichteten Verbindungen vorliegen, kennt hier auch Sensor j den Sensor i. Da im weiteren Verlauf die Veränderung oder das Entfernen der Sensoren gefordert wird, ist auch hier die Realisierung als 2-dimensionale doppelt verkettete Liste sinnvoll, da so Sensoren gelöscht werden können, ohne die gesamte Matrix zu restrukturieren.

Die Sensorwerte und die Verbindungen der Sensoren reichen dann aus, das Integral der Approximation des Sensornetzwerkes der Zählfunktion zu berechnen. Es werden keinerlei Informationen über die Koordinaten benötigt. Die Eingabedaten der nummerischen Berechnung können somit dargestellt werden, und die eigentliche Berechnung auf dieser Basis aufgebaut werden.

Nach *Theorem 4.3* gilt $\int_{\mathbb{R}^2} h\, d\chi = \sum_{s=0}^{\infty}(\beta_0\{h > s\} - \beta_0\{h \leq s\} + 1)$, somit muss für jedes s der Natürlichen Zahlen inclusive der Null die Nulle Betti-Zahl der jeweiligen Höhensätze berechnet werden. Es wird also die Anzahl der Zusammenhangskomponenten von Sensoren, dessen Messwerte größer, kleiner , gleich usw. dem Wert s sind, also die Anzahl der Zusammenhangskomponenten der Höhensätze bezüglich s gesucht. So wird man nach der Anzahl der Ansammlungen verbundener Sensoren mit Werten entsprechend s und des Höhensatzes gesucht.

Die Koordinatenfreiheit erlaubt es die Approximationsstruktur als Graph (Struktur in der Informatik) zu interpretieren. Als Knoten mit Werten und Verbindungen zwischen diesen Knoten. Die Nullte Betti Zahl der Level-Sets lässt sich dann einfach als Zusammenhangskomponente der Knoten mit Werten entsprechend s und der Höhenebene berechnen. Der Algorithmus basiert auf der Markierbarkeit der Knoten und ist eine Abwandlung der standardisierte Graphen Algorithmen (vgl. *Dijkstra Algorithmus/ Floyd-Warshall-Algorithmus*). Es werden alle Knoten als nicht markiert gekennzeichnet. Man betrachtet iterativ die Menge der nicht markierten Punkte, wählt einen aus, markiert ihn und erhöht zudem einen Counter um 1 falls die Bedingung zum Knotenwert passt und markiert solange alle Knoten, die einen Zusammenhang zur der in dieser Iteration markierten Menge haben und wieder zu der Bedingung passen, bis es keine Punkte mehr gibt, die zu dieser Bedingung passen und es schließlich auch keine nichtmarkierten Punkte mehr gibt. Somit erhält

man durch das Erhöhens eines Counters ausschließlich bei jedem Eintritt in die Verbindungssuche einer neuen Zusammenhangskomponente mit den passenden Knotenwerten die Anzahl der Zusammenhangskomponenten, also der gesuchten Anzahl der Zusammenhangskomponenten des Höhensatzes.

Der Algorithmus wird noch vereinfacht werden und gewisse Berechnungen zusammengelegt werden, sowie Speicherplatz und Geschwindigkeit gewonnen werden, doch vorerst liefert *Theorem 4.3* eine unendliche Summe. Dies ist aus nummerischer Sichtweise ein Problem, da die Berechnung nicht zu einem Ende kommen würde.

2.2.2 Abbruchbedingung

Da ein diskretes Sensornetzwerk aber nur endlich viele Sensoren hat, und unter der Annahme, dass die Messwerte der Zählsensoren nicht unendlich annehmen können, und das Sensornetzwerk genügend Konnektivität aufweist, ergeben die Summanden der Summe aus *Theorem 4.3* ab s gleich dem maximalen Messwert einfach Null.

$\sum_{s=Max(Sensorwerte)}^{\infty}(\beta_0\{h > s\} - \beta_0\{h \leq s\} + 1) = \sum_{s=Max(Sensorwerte)}^{\infty}(0 - 1 + 1) = 0$

Da $\beta_0\{h > s\} = 0$ für $s \geq Max(Sensorwerte)$, da es keine Sensoren mit Sensorwerten echt größer dem Maximum der Sensorwerte gibt, gibt es auch keine Zusammenhangskomponenten mit Werten echt größer s. Und weil alle Werte kleiner als s sind, und die Knoten sich alle untereinander wenigstens indirekt kennen, so ergibt $\beta_0\{h \leq s\} = 1$ für $s \geq Max(Sensorwerte)$, da ja die Zusammenhangskomponente dann alle Knoten umfasst, es befinden sich also alle Knoten in der gleichen Zusammenhangskomponente. Dies gilt es im Folgenden zu formalisieren, um darauf den Algorithmus aufzubauen.

Sei $S_n(W) \in W$ eine Menge der Koordinaten von n Sensoren in einem beschränkten, aber kontinuierlichen Zielraum $W \in \mathbb{R}^2$. Die n Sensoren sind dabei nummeriert, es existiert also eine festgelegte bijektive Funktion $s_n: \{1, \dots, n\} \to S_n(W)$, die jedem Sensor eine eindeutige natürliche Zahl zwischen 1 und n zugeteilt. Demnach sei $A_n \in \{0,1\}^{n \times n}$ die Adjazenz Matrix der Sensorverbindungen bezüglich der Zuweisungsfunktion s_n. Dabei kennt ein Sensor i einen Sensor j, falls $A_n(i,j) = 1$, andernfalls kennt Sensor i Sensor j nicht. Jeder Knoten kennt sich selbst. Auf dieser Basis kann man dann definieren:

2.2.2.1 Definition: zusammenhängendes Sensornetzwerk

Eine endliche Sensormenge $S_n(W)$ eines beschränkten, aber kontinuierlichen Zielraum W ist in Bezug auf s_n und $A_n \in \{0,1\}^{n\times n}$, der Adjazenz Matrix der Sensorverbindungen, zusammenhängend, falls für alle $(i,j) \in \{1,\dots,n\}^2$ eine endliche Folge mit $m \in \{1,\dots,n\}$ Folgengliedern, $\left(a_{i,j}\right)_{\{1,\dots,m\}} \in \{1,\dots,n\}$, mir Werten zwischen 1 und n existiert, sodass das erste Folgenglied den Wert i annimmt, also $a_{i,j}(1) = i$, das letzte Folgenglied den Wert j annimmt, $a_{i,j}(m) = j$, sowie jeder Eintrag $A_n\left(a_{i,j}(k), a_{i,j}(k+1)\right) = 1$ für jedes $k \in \{1,\dots,m-1\}$. Dies bedeutet, dass alle Sensoren in $S_n(W)$ in Verbindung stehen.

Beliebige Sensoren des Sensorraumes $S_n(W)$ sind also immer miteinander verbunden, wenn auch über Umwege. Dies ist leicht einzusehen. $A_n(Sensornummer\ a, Sensornummer\ b) = 1$ bedeutet, Sensor a kennt Sensor b. Folglich liefert $A_n\left(a_{i,j}(k), a_{i,j}(k+1)\right) = 1$, jeder Sensor mit Sensornummer $a_{i,j}(k)$ kennt den Sensor mit Sensornummer des nächsten Folgengliedes $a_{i,j}(k+1)$. Sensor $a_{i,j}(1)$ kennt also $a_{i,j}(m)$, da Sensor $a_{i,j}(1)$ kennt Sensor $a_{i,j}(2)$, dieser kennt $a_{i,j}(3)$, … und Sensor $a_{i,j}(m-1)$ kennt Sensor $a_{i,j}(m)$. Mit $a_{i,j}(1) = i$ und $a_{i,j}(m) = j$ kennt Sensor i also Sensor j. Existiert also für alle Paare $(i,j) \in \{1,\dots,n\}^n$ solch eine Folge, so kennen sich alle Sensoren des Sensornetzwerkes indirekt.

Wenn ein Zielaufzählproblem gestellt wird, ist eine logische Grundvoraussetzung des (endlichen) Erfolges eine endliche Anzahl an Zielen, insbesondere an Zielträgern. Darauf aufbauend kann also kein Sensor, der seine Zählfunktion richtig ausführt, in diesem Modell mehr als eine endliche Anzahl an Zielträgern messen.

2.2.2.2 Definition: gültige Sensormesswerte

Für eine endliche Sensormenge $S_n(W)$ eines beschränkten, aber kontinuierlichen Zielraum W und eine endliche Menge Zielträger $\{U_i\}$ von Zielen $\{O_i\}$ in W liefert $S_n(W)$ eine Zählfunktion $h_n: S_n(W) \to \mathbb{N}_0$, die für jeden Sensor die Anzahl der erkannten Zielträger zurückgibt. Diese Zählfunktion $h_n$nimmt für keinen Sensor aus $S_n(W)$ einen Wert größer der Anzahl der Zielträger $\{U_i\}$ an, also insbesondere einen endlichen Wert. Liegt eine solche Funktion h_n vor, so liefert h_n gültige Sensormesswerte.

Beweis:

Angenommen, die Zählfunktion h_n nimmt bei Sensor $u \in S_n(W)$ den Wert $h_n(u) > \#\{U_i\}$ an, so werden $h_n(u)$ Zielträger durch Sensor m erkannt. Es gibt aber nach Voraussetzung nur $\#\{U_i\} < \infty$ Zielträger. ■

2.2.2.3 Definition: Zusammenhangskomponente der Höhensätze (Nullte Betti Zahl)

Sei $S_n(W)$ eine endliche Sensormenge eines beschränkten, aber kontinuierlichen Zielraum W. Sei $A_n \in \{0,1\}^{n\times n}$ die zu der Zuordnungsfunktion s_n gehörige Adjazenz Matrix der Sensorverbindungen. Für eine endliche Menge Zielträger $\{U_i\}$ von Zielen $\{O_i\}$ in W liefert $S_n(W)$ eine Zählfunktion $h_n: S_n(W) \to \mathbb{N}_0$ mit gültigen Sensormesswerten. Sei $s \in \mathbb{N}_0$. Die nullte Betti Funktion

$$\beta_{0,n}(\, S_n(W)\,, h_n, A_n\,, s\,, \{<, >, \leq, \geq, =\}) \to \mathbb{N}_0$$

der Höhensätze h_n $[<, >, \leq, \geq, =]$ s entspricht in dieser Interpretation folgendem:

Sei $B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n) = \{\, x \in s_n^{-1}(S_n(W)) \mid h_n(s_n(x))\ [<, >, \leq, \geq, =]\ s \,\}$ die Menge aller Sensornummern, dessen zugehörige Sensoren Werte $[<, >, \leq, \geq, =]$ s messen. Dabei soll die Notation $[<, >, \leq, \geq, =]$ die Möglichkeiten der Vergleichsfälle zusammenfassen, mit der B definiert werden kann.

Für jede Sensornummer $x \in B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n)$ definiere

$$Z_x\big(B_{s,\{<,>,\leq,\geq,=\}}(S_n(W)), A_n\big) =$$

$$\{\, i \in B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n) \mid \exists\, m \in \{1, \dots, n\}\,,$$
$$\text{und eine Folge } (z_n)_{\{1,\dots,m\}} \in B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n),\ \text{sodass } z_n(1) = x,$$
$$z_n(m) = i \text{ und } A_n\big(z_n(r), z_n(r+1)\big) = 1 \text{ für alle } r \in \{1, \dots, m-1\}\,\}$$

, die Sensornummermenge der Sensoren in $B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n)$, die von dem Sensor mit Nummer x aus ausschließlich über Sensoren aus der Sensornummermenge $B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n)$ erreicht werden können.

$\beta_{0,n}(\, S_n(W)\,, h_n, A_n\,, s\,, \{<, >, \leq, \geq, =\})$ entspricht der Anzahl der verschiedenen Zusammenhangskomponenten der Menge $B_{s,\{<,>,\leq,\geq,=\}}(S_n(W))$, also der Anzahl unterschiedlicher Mengen $Z_x\big(B_{s,\{<,>,\leq,\geq,=\}}(S_n(W)), A_n\big)$ für $x \in B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n)$:

$$\beta_{0,n}(\, S_n(W)\,, h_n, A_n\,, s\,, \{<, >, \leq, \geq, =\}) =$$

$$\#\{Z_x\big(B_{s,\{<,>,\leq,\geq,=\}}(S_n(W)), A_n\big) \mid x \in B_{s,\{<,>,\leq,\geq,=\}}(S_n(W), h_n)\,\}$$

Dabei darf die Mächtigkeit der Menge nicht im Sinne einer Multimenge gesehen werden. Mengen, die mehrfach vorkommen, werden nur einmal gezählt.

Mit diesen Definitionen kann nun eine Abbruchbedingung der Summe aus *Theorem 4.3* aufgestellt werden und getestet werden, ob ein Sensornetzwerk zusammenhängend ist. Um die Formulierungen zu vereinfachen, wird die Notation

$\beta_{0,n,A_n}([h_n\ [<,>,\leq,\geq,=]\ s]) =: \beta_{0,n}(S_n(W), h_n, A_n, s, \{<,>,\leq,\geq,=\})$ eingeführt.

2.2.2.4 Proposition: Kriterium eines zusammenhängenden Sensornetzwerkes

Sei $S_n(W)$ eine endliche Sensormenge eines beschränkten, aber kontinuierlichen Zielraum W. Sei $A_n \in \{0,1\}^{n\times n}$ die zu der Zuordnungsfunktion s_n gehörige Adjazenz Matrix der Sensorverbindungen. Sei zu $S_n(W)$ eine Zählfunktion $h_n: S_n(W) \to \mathbb{N}_0$ mit gültigen Sensormesswerten gegeben. Sei β_{0,n,A_n} wie in Definition 2.2.2.3 nach der genannten Notation.

Falls $\beta_{0,n,A_n}([h_n \leq s]) = 1$ für $s = max\{h_n(S_n(W))\}$, so liefert die Adjazenz Matrix A_n ein zusammenhängendes Sensornetzwerk.

Beweis:

Sei $s = max\{h_n(S_n(W))\}$. *Dies ist auf Grund der gültigen Sensormesswerte und der endlichen Sensoranzahl möglich. Alle Sensoren haben somit Sensormesswerte kleiner oder gleich s. Somit berechnet $\beta_{0,n,A_n}([h_n \leq s])$ die Anzahl der Zusammenhangskomponenten des Sensornetzwerkes, da jeder Sensor auf die Bedingung $h_n \leq s$ zutrifft. Gilt somit $\beta_{0,n,A_n}([h_n \leq s]) = 1$, so wird allen Sensornummern $x \in s_n^{-1}(S_n(W))$ die Gleiche Menge $Z_x(B_{s,\{\leq\}}(S_n(W)), A_n)$ nach Bezeichnungen der Definition 2.2.2.3 zugeordnet, nämlich schon die Gesamte Menge $s_n^{-1}(S_n(W))$, also die Menge aller Sensornummern des Sensornetzwerkes $S_n(W)$. Jeder Sensor kennt somit gesamt $S_n(W)$ indirekt. Somit ist das Sensornetzwerk $S_n(W)$ unter A_n zusammenhängend nach Definition 2.2.2.1.* ■

2.2.2.5 Proposition: Endlicher Abbruch der Summe in *Theorem 4.3.*

Sei $S_n(W)$ eine endliche Sensormenge eines beschränkten, aber kontinuierlichen Zielraum W. Sei $A_n \in \{0,1\}^{n\times n}$ die zu der Zuordnungsfunktion s_n gehörige zusammenhängende Adjazenz Matrix der Sensorverbindungen, im Sinne der Definition 2.2.2.1. Sei zu $S_n(W)$ eine Zählfunktion $h_n: S_n(W) \to \mathbb{N}_0$ mit gültigen Sensormesswerten gegeben. Sei β_{0,n,A_n} wie in Definition 2.2.2.3 nach der genannten Notation.

Sei $s \geq max\{h_n(S_n(W))\}$, *so gilt* $\beta_{0,n,A_n}([h_n \leq s]) = 1$ *und* $\beta_{0,n,A_n}([h_n > s]) = 0$. *Insbesondere folgt aus* $\beta_{0,n,A_n}([h_n > s]) = 0$ *auch, dass* $s \geq max\{h_n(S_n(W))\}$.

Beweis:

„=>“:

Sei $s \geq max\{h_n(S_n(W))\}$. *Solch ein s kann auf Grund der gültigen Sensormesswerte und der endlichen Sensoranzahl gefunden werden. Alle Sensoren haben somit Sensormesswerte kleiner oder gleich s. Somit berechnet* $\beta_{0,n,A_n}([h_n \leq s])$ *die Anzahl der Zusammenhangskomponenten des Sensornetzwerkes, da jeder Sensor auf die Bedingung* $h_n \leq s$ *zutrifft, also einen Sensorwert kleiner gleich s aufweist. Somit gilt* $\beta_{0,n,A_n}([h_n \leq s]) = 1$, *da das Sensornetzwerk zusammenhängend ist, und somit die Mengen* $Z_x(B_{s,\{\leq\}}(S_n(W)), A_n)$ *[nach Bezeichnungen der Definition 2.2.2.3] der Sensoren mit Nummern* $x \in s_n^{-1}(S_n(W))$ *alle bereits gleich der Menge* $\{1,\ldots,n\} = s_n^{-1}(S_n(W))$, *der Menge aller Sensornummern, sind.*

Sei wieder $s \geq max\{h_n(S_n(W))\}$, *alle Sensoren haben somit Sensormesswerte kleiner oder gleich s. Die Menge* $B_{s,\{>\}}(S_n(W), h_n)$ *[Definition 2.2.2.3] ist somit leer, da kein Sensorwert die Bedingung erfüllen, echt größer als s zu sein. Damit hat die Menge* $B_{s,\{>\}}(S_n(W), h_n)$ *[Definition 2.2.2.3] auch keine Teilmengen* $Z_x(B_{s,\{>\}}(S_n(W)), A_n)$. *Somit gilt* $\beta_{0,n,A_n}([h_n > s]) = 0$.

„<=“:

Sei $\beta_{0,n,A_n}([h_n > s]) = 0$. *Dies Bedeutet, die Menge* $B_{s,\{>\}}(S_n(W), h_n)$ *[Definition 2.2.2.3] ist leer, da für* $x \in B_{s,\{>\}}(S_n(W), h_n)$ *auch gelten würde* $x \in Z_x(B_{s,\{>\}}(S_n(W)), A_n)$, *somit wäre also* $Z_x(B_{s,\{>\}}(S_n(W)), A_n)$ *nicht leer , und damit* $\beta_{0,n,A_n}([h_n > s]) \neq 0$. *Somit gibt es nach der Definition von* $B_{s,\{>\}}(S_n(W), h_n)$ *[Definition 2.2.2.3] keinen Sensorwert echt größer s, also* $s \geq max\{h_n(S_n(W))\}$. ■

Somit kann die Summe bei dem ersten s mit $\beta_0([h_n > s]) = 0$ abgebrochen werden, da so der maximale Sensormesswert erreicht wurde, und die die weitere Summe wie oben dargelegt immer zu Null aufaddiert. Nun kann mittels dieser Grundlage *Theorem 4.3* als Algorithmus umgesetzt werden.

2.2.3 Umsetzung des Theorems 4.3 via Adjazenz Matrix (in Python 3)

Funktion: euler_calcus_2d

In:

```
Knotenwerte; Liste mit Knotenwerten,
    Knoten i hat den Knotenwert: (Knotenwerte[i])
Knotenverbindungen; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
    (Knotenverbindungen[i,j] == 1)
```

Out:

```
Integer; Berechnetes Ergebnis der oben erwähnten Formel anhand der Knotenwerte
    und ihrer Verbindungen
```

Kommentar:

z2: Laufindex, entspricht dem s aus der Summe des Theorems 4.3

z3: Berechnet Anzahlen der Zusammenhangskomponenten für s = 0 und speichert sie in einer Variable, um doppelte Berechnungen zu vermeiden.

z4: Abbruchbedingung: Wenn es keine Knoten mehr gibt dessen Knotenwert größer als der Laufindex s ist

```
z5: Umsetzen der Berechnungsformel:

    beta_0(..,s)[0] gibt die Anzahl der Zusammenhangskomponenten
    echt größer s zurück

    beta_0(..,s)[1] gibt die Anzahl der Zusammenhangskomponenten
    kleiner gleich s zurück


z6: Den Laufindex erhöhen

z7: Aktualisiert die Anzahlen der Zusammenhangskomponenten für die nächste
    Iteration zum vermeiden mehrfacher Berechnung
```

z8: Überprüfung; War das Gitter zusammenhängend:

```
    Falls (beta_0(knotenwerte,knotenverbindungen,s)[1] ungleich 1) für
    (s = max(Knotenwerten)) so was das Gitter nicht zusammenhängend, denn
    falls doch, wären ja die    Knotenwerte der Knoten alle kleiner gleich s
    und somit alle Knoten in einer Zusammenhangskomponente, also gleich 1.
```

```python
def euler_calcus_2d(knotenwerte,knotenverbindungen):
    summe = 0
    s = 0
    temp = beta_0(knotenwerte,knotenverbindungen,s)
    while(temp[0] !=0):
        summe =  summe + ( temp[0] - temp[1] + 1)
        s = s + 1
        temp = beta_0(knotenwerte,knotenverbindungen,s)
    if(temp[1] != 1):
        summe = 0
        print("Gitter war nicht zusammenhängend")
    return summe
```

Funktion: beta_0

In:

```
Knotenwerte; Liste mit Knotenwerten,
    Knoten i hat den Knotenwert: (Knotenwerte[i])
Knotenverbindungen; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
    (Knotenverbindungen[i,j] == 1)
S; Laufindex; Integer, Bedeutung siehe unten.
```

Out:

```
2 Integer; Gibt Erstens die Anzahl der Verbindungskomponenten dessen Knotenwerte
echt größer als s sind zurück, Zweitens die Anzahl der Verbindungskomponenten dessen Knoten
werte
kleiner gleich als s sind
```

Kommentar:

z1: Counter der Zusammenhangskomponenten echt größer als s

z2: Counter der Zusammenhangskomponenten kleiner gleich als s

z3/4/5: Legt eine Liste mit einem Listenelement pro Knoten an, das mit 0 belegt wird. Diese Liste dient zum Markieren der Knoten, damit man sie nicht erneut aufsucht

z6: Durchlauf aller Knoten:

```
z7: Falls Knoten noch nicht markiert, und Knotenwert des Knotens echt größer s:

    z8: Erhöhe den Counter für echt größer um eins
    z9: Markiere den Knoten
    z10: Markiere alle Knoten die mit dem Knoten oder der entstehenden
        Zusammenhangskomponente zusammenhängen und den Knotenwert
        echt größer als s haben

z11: Falls anderseits Knoten nicht markiert und Knotenwert kleiner gleich s:

    z12: Erhöhe den Counter für kleiner gleich um eins
    z13: Knoten Markieren
    z14: Markiere alle Knoten die mit dem Knoten oder der entstehenden
         Zusammenhangskomponente zusammenhängen und den Knotenwert
         kleiner gleich als s haben
z15: Gibt die Counter zurück, also erstens die Anzahl der Zusammenhangskomponenten
     mit Werten echt größer als s und zweitens die kleiner gleich s.
```

```python
def beta_0(knotenwerte,knotenverbindungen,s):
    counter_echtgr = 0
    counter_kl = 0
    marker = []
    for i in range (0,len(knotenverbindungen)):
        marker = marker + [0]
    for j in range (0,len(knotenverbindungen)):
        if(marker[j] == 0 and knotenwerte[j] > s):
            counter_echtgr = counter_echtgr + 1
            marker[j] = 1
            marker = verbindungssuche_echtgr(knotenwerte,knotenverbindungen,s,marker,j)
        elif(marker[j] == 0 and knotenwerte[j] <= s):
            counter_kl = counter_kl + 1
            marker[j] = 1
            marker = verbindungssuche_kl(knotenwerte,knotenverbindungen,s,marker,j)
    return counter_echtgr,counter_kl
```

Funktion: verbindungssuche_echtgr

In:

```
Knotenwerte; Liste mit Knotenwerten,
           Knoten i hat den Knotenwert: (Knotenwerte[i])
Knotenverbindungen; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
           (Knotenverbindungen[i,j] == 1)
S; Laufindex; Integer
J; Knotennummer; Integer, Knotennummer des Knotens von dem aus
           die Nachbarschaft überprüft werden soll
```

Out:

```
Liste; Markierliste, in der alle besuchten Knoten markiert sind, die in
            derselben Zusammenhangskomponente echt größer s liegen,
            wie der Knoten mit der Knotennummer j.
```

Kommentar:

z1: Läuft einmal mit i durch die Verbindungen des Knotens mit Nummer j, also durch die Zeile[j] der Knotenverbindungen

```
z2: Falls Knoten j Knoten i kennt, und außerdem der Knotenwert von i echt größer s ist
    und der Knoten unmarkiert ist:

    z3: Markiere den Knoten
    z4: Markiere rekursiv alle Knoten die in Verbindung zu Knoten i oder
        der gesamten Zusammenhangskomponente stehen und dessen Knotenwert echt größer als s
sind.
```

z5: Siehe oben bei 'Out'.

```python
def verbindungssuche_echtgr(knotenwerte,knotenverbindungen,s,marker,j):
    for i in range (0,len(knotenverbindungen)):
        if(knotenverbindungen[i][j] == 1 and knotenwerte[i] > s and marker[i] == 0):
            marker[i] = 1
            marker = verbindungssuche_echtgr(knotenwerte,knotenverbindungen,s,marker,i)
    return marker
```

Funktion: verbindungssuche_kl

In:

```
Knotenwerte; Liste mit Knotenwerten,
           Knoten i hat den Knotenwert: (Knotenwerte[i])
Knotenverbindungen; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
           (Knotenverbindungen[i,j] == 1)
S; Laufindex; Integer
J; Knotennummer; Integer, Knotennummer des Knotens von dem aus
           die Nachtbarschaft überprüft werden soll
```

Out:

```
Liste; Markierliste, in der alle besuchten Knoten markiert sind, die in
            derselben Zusammenhangskomponente kleiner gleich s liegen,
            wie der Knoten mit der Knotennummer j.
```

Kommentar:

z1: Läuft einmal mit i durch die Verbindungen des Knotens mit Nummer j, also durch die Zeile[j] der Knotenverbindungen

```
z2: Falls Knoten j Knoten i kennt, und außerdem der Knotenwert von i kleiner gleich s ist
    und der Knoten unmarkiert ist:

    z3: Markiere den Knoten
    z4: Markiere rekursiv alle Knoten die in Verbindung zu Knoten i oder
        der gesamten Zusammenhangskomponente haben und dessen Knotenwert kleiner gleich s i
st.
```

z5: Siehe oben bei 'Out'.

```python
def verbindungssuche_kl(knotenwerte,knotenverbindungen,s,marker,j):
    for i in range (0,len(knotenverbindungen)):
        if(knotenverbindungen[i][j] == 1 and knotenwerte[i] <= s and marker[i] == 0):
            marker[i] = 1
            marker = verbindungssuche_kl(knotenwerte,knotenverbindungen,s,marker,i)
    return marker
```

2.2.4 Verbesserung über Adjazenz Listen

Die Nullwerte der Adjazenz Matrix benötigen Speicherplatz und vertreten Informationen, die erstens schon indirekt durch die Einsen repräsentiert werden, und zweitens in dem Algorithmus nicht benötigt werden. Zudem benötigt der Algorithmus viele Abfragen, um die Verbindungsinformation eines einzelnen Sensors zu entnehmen. Nullwerte in der Adjazenz Matrix sind häufig stark vertreten, da ein Knoten aus physikalischen Gründen meist nur mit einer im Verhältnis zur Gesamtzahl der Sensoren kleinen Anzahl der gesamten Sensoren in Kontakt steht. Das führt auf die Idee, zu jedem Sensor nur die Sensornummer zu speichern, die der Sensor direkt kennt. Solch eine Adjazenz Liste speichert also zu jeder Knotennummer die direkt verbundenen Knotennummern. Damit wird nicht nur Speicherplatz eingespart, sondern auch die Geschwindigkeit des Algorithmus auf folgende Weise verbessert:

Da sich der Algorithmus sonst kaum verändert sind nur die wichtigen Funktionen angegeben, die sich bei Nutzung der Adjazenz Liste verändern. Das vollständige Programm befindet sie in dem Programmteil.

Funktion: verbindungssuche_echtgr_ad

In:

```
Knotenwerte; Liste mit Knotenwerten,
           Knoten i hat den Knotenwert: (Knotenwerte[i])
Adjazenz Liste; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
    i in Adjazenz Liste[j] und j in Adjazenz Liste[i]
S; Laufindex; Integer
J; Knotennummer; Integer, Knotennummer des Knotens von dem aus
           die Nachtbarschaft überprüft werden soll
```

Out:

```
  Liste; Markierliste, in der alle besuchten Knoten markiert sind, die
in
             derselben Zusammenhangskomponente echt größer s liegen,
             wie der Knoten mit der Knotennummer j.
```

Kommentar:

z1: Läuft einmal mit i durch die verbundenen Knoten des Knotens mit Nummer j, also durch die Zeile[j] der Adjazenz Liste

```
  z2: Falls der Knotenwert von Knoten i echt größer s ist und der Knote
n i unmarkiert ist:

      z3: Markiere den Knoten
      z4: Markiere rekursiv alle Knoten die in Verbindung zu Knoten i o
der
          der gesamten Zusammenhangskomponente stehen und dessen Knoten
wert echt größer als s sind.
```

z5: Siehe oben bei 'Out'.

```
def verbindungssuche_echtgr_ad(knotenwerte,adjazenzliste,s,marker,j):
    temp = []
    for i in adjazenzliste[j]:
        if(knotenwerte[i] > s and marker[i] == 0):
            marker[i] = 1
            temp = temp + [i]
            marker = verbindungssuche_echtgr_ad(knotenwerte,adjazenzliste,s,ma
rker,i)
    return marker
```

Funktion: verbindungssuche_kl_ad

In:

```
Knotenwerte; Liste mit Knotenwerten,
           Knoten i hat den Knotenwert: (Knotenwerte[i])
Adjazenz Liste; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
    i in Adjazenz Liste[j] und j in Adjazenz Liste[i]
S; Laufindex; Integer
J; Knotennummer; Integer, Knotennummer des Knotens von dem aus
           die Nachtbarschaft überprüft werden soll
```

Out:

```
  Liste; Markierliste, in der alle besuchten Knoten markiert sind, die
in
             derselben Zusammenhangskomponente kleiner gleich s liege
n,
             wie der Knoten mit der Knotennummer j.
```

Kommentar:

z1: Läuft einmal mit i durch die verbundenen Knoten des Knotens mit Nummer j, also durch die Zeile[j] der Adjazenz Liste

```
  z2: Falls der Knotenwert von Knoten i kleiner gleich s ist und der Kn
oten i
      unmarkiert ist:

      z3: Markiere den Knoten
      z4: Markiere rekursiv alle Knoten die in Verbindung zu Knoten i o
der
          der gesamten Zusammenhangskomponente haben und dessen Knotenw
ert kleiner gleich s ist.
```

z5: Siehe oben bei 'Out'.

```
def verbindungssuche_kl_ad(knotenwerte,adjazenzliste,s,marker,j):
    temp = []
    for i in adjazenzliste[j]:
        if(knotenwerte[i] <= s and marker[i] == 0):
            marker[i] = 1
            temp = temp + [i]
            marker = verbindungssuche_kl_ad(knotenwerte,adjazenzliste,s,marke
r,i)
    return marker
```

2.3 Minimaler Charakterisiergraph

2.3.1 Idee der Minimalisierung

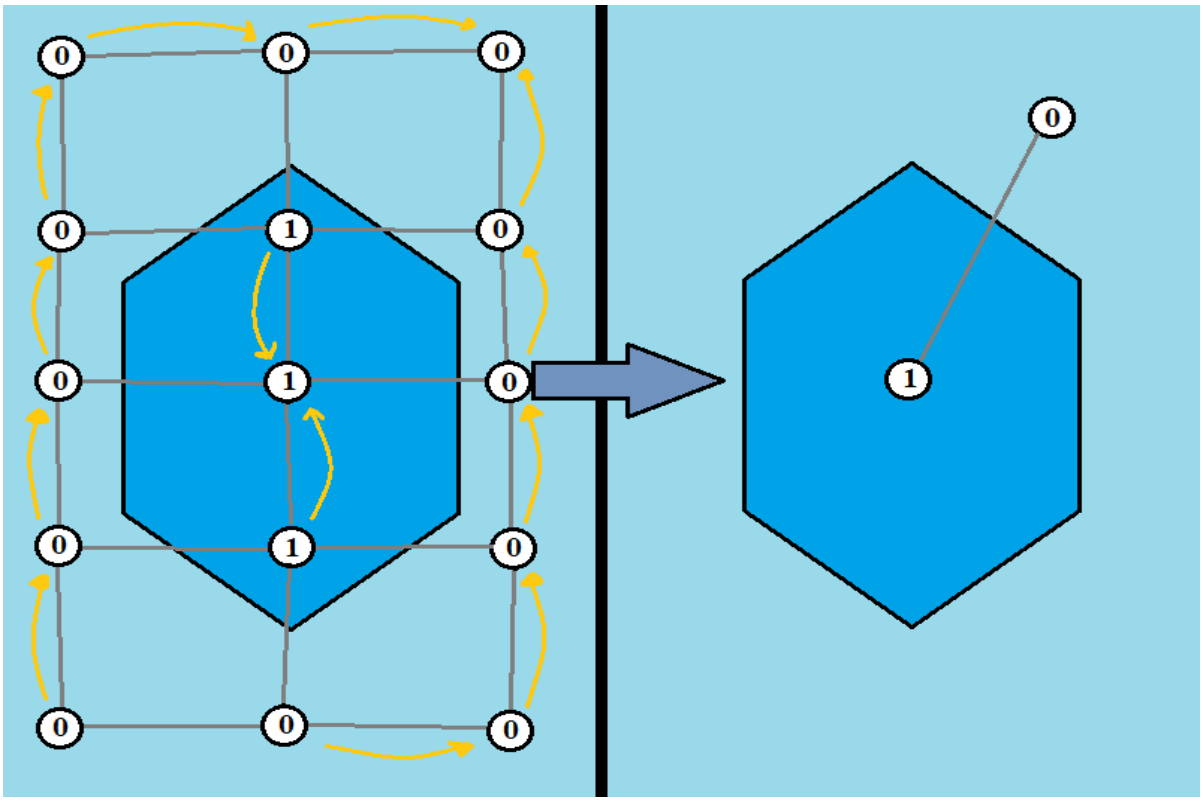

Abbildung 7: Ein Sechseck wird durch ein diskretes Netzwerk approximiert. Die Knoten innerhalb einer Zusammenhangskomponente können nun zusammengezogen werden, wie mit den gelben Pfeilen angedeutet wird.

Ähnlich den Ideen aus der Veröffentlichung von Tanaka (Tanaka, 2017), nur in diesem Fall entlang des Algorithmus entwickelt, existieren angenommen, ein Sensornetzwerk reiche aus, um das Euler Integral einer gegebenen Struktur genau zu berechnen, für die Berechnung oft überflüssige Knoten. Nämlich genau diese, die sich in einer Zusammenhangskomponente mit Sensoren gleicher Sensorwerte befinden (Abbildung 7). Zieht man diese Sensoren zu einem zusammen, und stellt die Verbindungen, die die Verbindungskomponente mit anderen Sensoren hatte, wieder her, so ändert sich der über *Theorem 4.3* berechnete Wert nicht. Dies kann auch für die realisierten Algorithmus der Zusammenziehung der Sensoren in 2.3.2 über die Schleifeninvariante bewiesen werden.

Theorem 4.3 berechnet sich über die nullten Betti Funktionen spezieller Höhensätze. Die Berechnung ändert sich nicht, wenn die nullten Betti Funktionen aller Höhensätze gleiche Werte für die Zusammengezogene Struktur (m_n, k_n), wie für das Ausgangsnetzwerk (A_n, K_n) liefern. Sei $s \in \mathbb{N}_0$beliebig. Dabei ist m_n die Verbindungsmatrix der Sensorverbindungen mit Sensorwerten k_n der Zusammengezogenen Struktur, A_n die Verbindungsmatrix der Sensorverbindungen mit Sensorwerten K_n der ausgangs Struktur. Die Funktion $\beta_{0,n,A_n}([h_n =$

s]) berechnet die Anzahl der Zusammenhangskomponenten von Sensoren mit Sensorwerten gleich s. Der Zusammenziehende Algorithmus aus 2.3.2 zieht jeden Sensor mit zugehörigem Sensorwert, der noch nicht zusammengezogen wurde, mit Sensoren gleicher Sensorwerte, die untereinander miteinander verbunden sind, zusammen, und stellt die Verbindungen wieder her. Jeder Sensor, der mit der Komponente verbunden war, ist nun mit dem neuen Repräsentanten der Komponente verbunden, alle anderen Sensoren und Verbindungen werden gelöscht. Die Schleifeninvariante besagt nun, die Funktion $\beta_{0,n,m_n^i}([h_n = s])$ berechnet den gleichen Wert wie die Funktion $\beta_{0,n,m_n^{i+1}}([h_n = s])$ nach Zusammenziehen der Zusammenhangskomponente im i-ten Schleifendurchlauf. Angenommen die Schleifeninvariante gilt vor dem i-ten Schritt. Der Zusammenziehende Algorithmus berechnet im i-ten Schritt die Sensoren, die eine Zusammenhangskomponente gleicher Knotenwerte mit dem im Schritt i betrachteten Sensor bilden, und löscht alle Sensoren bis auf den Sensor i, und stellt die Verbindungen wie oben genannt wieder her. Die Funktion $\beta_{0,n,m_n^i}([h_n = s])$ berechnet die Anzahl der Zusammenhangskomponenten von Sensoren mit Sensorwerten gleich s. Die Anzahl der Zusammenhangskomponenten wurde aber nicht verändert, da der Repräsentant existiert, somit die Verbindungen inclusive der Anzahl erhält. Es können durch das zusammenziehen keine Zusammenhangskomponenten zusammenfallen oder entstehen, da nur bereits existierende Verbindungen übernommen werden, und würden zwei Komponenten in der weiteren Berechnung nochmal zusammenfallen, so hätten sie schon bei der Berechnung ihrer Entstehung zusammenfallen müssen, da sie ja eine Verbindung zueinander hatten, da diese ja nicht neu entstehen kann. Somit gilt nach Zusammenfügen:

$\beta_{0,n,m_n^i}([h_n = s]) = \beta_{0,n,m_n^{i+1}}([h_n = s])$, somit ist die Schleifeninvariante bewiesen. Damit ändert der Zusammenziehende Algorithmus die Anzahl der Zusammenhangskomponenten gleicher Sensorwerte nicht. Die anderen Funktionen $\beta_{0,n,A_n}([h_n\ [<,>,\leq,\geq]\ s])$ berechnen nun die Anzahl der Zusammenhangskomponenten $[<,>,\leq,\geq]$ *s* der Zusammenhangskomponenten gleicher Sensorwerte. Da die Anzahl der Zusammenhangskomponenten gleicher Sensorwerte gleichbleibt, und der Repräsentant der Zusammenhangskomponente diese in Anzahl und Verbindung vertritt, wird auch die Anzahl der Zusammenhangskomponenten $[<,>,\leq,\geq]$ s gleichbleiben. Damit bleiben alle Nullten Betti Zahlen für beliebige s nach Zusammenführung gleich, also

$\beta_{0,n,A_n}([h_n = s]) = \beta_{0,n,m_n}([h_n = s])$, und damit bleibt auch die Berechnung in *Theorem 4.3* gleich.

Folgend die Umsetzung der Zusammenziehenden Algorithmus in Python, hier Minimales_Gitter genannt.

2.3.2 Programmtechnische Umsetzung (in Python 3)

Funktion: lokaler_Zusammenhang

In:

```
Knotenwerte; Liste mit Knotenwerten,
           Knoten i hat den Knotenwert: (Knotenwerte[i])
Adjazenz Liste; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
    i in Adjazenz Liste[j] und j in Adjazenz Liste[i]
Knotennummer; Integer, Knotennummer des Knotens von dem aus
           die Nachtbarschaft auf Knoten mit gleichem Knotenwert über
prüft werden soll
```

Out:

```
  Liste; Liste mit Knotennummern der Knoten die den gleichen Knotenwert
wie Knoten j haben
  und in seiner Zusammenhangskomponente liegen
```

Kommentar:

z1: Merkt sich den Knotenwert des Knotens 'knotennummer', um darauf die Nachtbarschaft auf Knoten diesen Wertes zu testen

z2: Legt eine leere Liste an, um darin die Nummern der Knoten zu speichern, die mit Knoten 'knotennummer' eine Zusammenhangskomponente gleicher Knotenwerte bilden

z3: Für jeden Nachbarn des Knotens 'knotennummer':

```
  z4: Falls die Knotenwert des Nachbarn übereinstimmt und die Knotennum
mer dieses Knotens nicht bereits
      in der Liste der Knoten, die in z2 angelegt wurde, ist:

      z5: Ergänze die Liste zusammenhängende_knoten um die Knotennummer
des Nachtbarn
      z6: Füge rekursiv alle weiteren Nachbarn der Zusammenhangskompone
nte hinzu
          die den gleichen Knotenwert haben:
          lokaler_Zusammenhang_recrusiv(..,,adjazensliste[knotennumme
r][i],zusammenhängende_knoten);
```

z7: Gebe die Liste zusammenhängende_knoten zurück.

```
def lokaler_Zusammenhang(knotenwerte,adjazensliste,knotennummer):
    knotenwert = knotenwerte[knotennummer]
    zusammenhängende_knoten = []
    for i in range(0,len(adjazensliste[knotennummer])):
        if(knotenwert == knotenwerte[adjazensliste[knotennummer][i]] and knotennummer not in zusamme
nhängende_knoten):
            zusammenhängende_knoten = zusammenhängende_knoten + [adjazensliste[knotennummer][i]]
            zusammenhängende_knoten = lokaler_Zusammenhang_recrusiv(knotenwerte,adjazensliste,adjaze
nsliste[knotennummer][i],zusammenhängende_knoten).copy()
    return zusammenhängende_knoten
```

Funktion: lokaler_Zusammenhang_recrusiv

In:

```
  Knotenwerte; Liste mit Knotenwerten,
               Knoten i hat den Knotenwert: (Knotenwerte[i])
  Adjazenz Liste; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
               i in Adjazenz Liste[j] und j in Adjazenz Liste[i]
  Knotennummer; Integer, Knotennummer des Knotens von dem aus
               die Nachbarschaft auf Knoten mit gleichem Knotenwert überp
rüft werden soll
  Zusammenhängende_knoten; Liste mit Knotennummern: Knotennummern der K
noten die bereits
               in der Zusammenhangskomponente enthalten sind, um so doppe
ltes hinzufügen zu vermeiden.
```

Out:

```
  Liste; Liste mit Knotennummern der Knoten die den gleichen Knotenwert
wie Knoten j haben
  und in seiner (vorerst) direkten Nachtbarschaft liegen
```

Kommentar:

z1: Merkt sich den Knotenwert des Knotens 'knotennummer', um darauf die Nachbarschaft auf Knoten diesen Wertes zu testen

z2: Laufe über alle Knoten, die der Knoten mit Nummer 'knotennummer' kennt, also durch Adjazenz Liste['knotennummer']:

```
  z3: Falls es nicht der Knoten mit der Knotennummer 'knotennummer' sel
bst ist
      und der Knotenwert des Nachbarn gleich ist und der Nachbar noch n
icht bereits enthalten ist:

      z4: Füge den Nachbarn hinzu
      z5: Rufe rekursiv die Suche für den Nachbarn auf, da er ja wieder
Nachbarn mit
          gleichem Knotenwert haben kann
```

z6: Gebe die Knotennummerliste zusammenhängende_knoten zurück.

```
def lokaler_Zusammenhang_recrusiv(knotenwerte,adjazensliste,knotennummer,zusammenhängende_knoten):
    knotenwert = knotenwerte[knotennummer]
    for i in range(0,len(adjazensliste[knotennummer])):
        if(adjazensliste[knotennummer][i] != knotennummer and knotenwert == knotenwerte[adjazenslist
e[knotennummer][i]] and adjazensliste[knotennummer][i] not in zusammenhängende_knoten):
            zusammenhängende_knoten = zusammenhängende_knoten + [adjazensliste[knotennummer][i]]
            zusammenhängende_knoten = lokaler_Zusammenhang_recrusiv(knotenwerte,adjazensliste,adjaze
nsliste[knotennummer][i],zusammenhängende_knoten).copy()
    return zusammenhängende_knoten
```

Funktion: Verbindungen_Zusammenziehen

In:

```
Adjazenz Matrix; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
            Adjazenz Matrix[i][j] == 1 == Adjazenz Matrix[j][i]
Knotenwerte; Liste mit Knotenwerten,
            Knoten i hat den Knotenwert: (Knotenwerte[i])
index_zusammenziehen; Liste, mit Knotennummmern der Knoten, die zusam
mengelegt werden sollen
```

Out:

```
Adjazenz Matrix(2d Liste);  Kürzer, mit zusammengelegten Knotenverbin
dungen.
```

Kommentar:

z1: Sortiert die Indexe der zusammenzulegenden Knoten, um die richtigen Knoten zu löschen. Andernfalls würde auf Grund der Listenstruktur der Matrix 'Adjazensmatrix' und des Feldes 'Knotenwerte' bei Löschungen innerhalb der Liste Indexverschiebungen hervorrufen und falsche Informationen entstehen. z2/3: Zusammenlegen aller Verbindungen der Knoten mit Nummer in index_zusammenziehen über die Funktion kenn_verknüpfung durch mehrfache Anwendung. Es wird eine neue Zeile erstellt, die des neue Knotens, der die Position der ersten Knotennummer in index_zusammenziehen annimmt.

z4-6: Trägt nach, welche Knoten Verbindungen zur alten Verbindungskomponente hatten, und somit auch den neuen Knoten kennen müssen.

z7-11: Nachdem keine Informationen mehr benötigt werden können die überflüssigen Zeilen und Spalteneinträge gelöscht werden. Ebenso werden die Knotenwerte der gelöschten Knoten entfernt. Dabei muss die Struktur der in Python realisierten Liste beachtet werden. Wenn man einen Knoten entfernt, so werden die Indizes der folgenden Knoten um eine Knotennummer reduziert, was bei weiterem Löschen (hinter!) dem Knoten beachtet werden muss. Deshalb ist die Liste auch vorsortiert.

z12: Gibt die kompaktere Adjazenz Matrix und die Knotenwertliste zurück

```
def Verbindungen_Zusammenziehen(adjazensmatrix,knotenwerte,index_zusammenziehen):
    index_zusammenziehen = sorted(index_zusammenziehen).copy()
    for i in index_zusammenziehen:
        adjazensmatrix[index_zusammenziehen[0]] = kenn_verknüpfung(adjazensmatrix[index_zusammenzieh
en[0]],adjazensmatrix[i]).copy()
    for l in range(0,len(adjazensmatrix)):
        if(adjazensmatrix[index_zusammenziehen[0]][l] == 1):
            adjazensmatrix[l][index_zusammenziehen[0]] = 1
    for j in range(1,len(index_zusammenziehen)):
        for k in range(0,len(adjazensmatrix)):
            del adjazensmatrix[k][index_zusammenziehen[j] - j + 1]
        del adjazensmatrix[index_zusammenziehen[j] - j + 1]
        del knotenwerte[index_zusammenziehen[j] - j + 1]
    return adjazensmatrix, knotenwerte
```

Funktion: kenn_verknüpfung

In:

```
  zeile_1; Liste, erste von zwei Zeilen der Adjazenz Matrix von zwei Kn
oten innerhalb einer
           Zusammenhangskomponente, dessen Verbindungen zusammengelegt
werden sollen
  zeile_2; Liste, zweite von zwei Zeilen der Adjazenz Matrix von zwei K
noten innerhalb einer
           Zusammenhangskomponente, dessen Verbindungen zusammengelegt
werden sollen
```

Out:

```
  Liste; Zusammengelegte Verbindungen der beiden Verbindungszeilen der
Adjazenz Matrix
```

Kommentar:

z1: Legt eine leere Zeile an, um sie aus den beiden Zeilen 'zeile_1' , 'zeile_2' zusammenzulegen

z2: Für i von 0 bis (Zeilenlänge der Adjazenz Matrix) -1 durchlaufe:

```
  z3: Falls einer der Einträge der beiden Zeile oder beide Eins sind, s
o:
      z4: Erweitere die Liste 'zeile' um einen Listeneintrag 1
  z3: Falls nicht, also sonst:
      z5: Erweitere die Liste 'zeile' um einen Listeneintrag 0
```

z6: Gebe die errechnete Zeile 'zeile' zurück.

```
def kenn_verknüpfung(zeile_1, zeile_2):
    zeile = []
    for i in range(0,len(zeile_1)):
        if(zeile_1[i] == 1 or zeile_2[i] == 1):
            zeile.append(1)
        else:
            zeile.append(0)
    return zeile
```

Funktion: Minimales_Gitter

In:

```
Adjazenz Matrix; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
          Adjazenz Matrix[i][j] == 1 == Adjazenz Matrix[j][i]
Knotenwerte; Liste mit Knotenwerten,
          Knoten i hat den Knotenwert: (Knotenwerte[i])
```

Out:

```
  Adjazenz Matrix; 2-dimensionale Liste, Adjazenz Matrix ohne Knoten di
e eine Zusammenhanskomponente
                im Sinne gleichen Knotenwerts echt größer eins haben.
  knotenwerte; Liste mit Knotenwerten, es wurden alle "überflüssigen" K
noten und deren Werte entfernt.
```

Kommentar:

z1: Setzt eine obere Grenze des Durchlaufs durch die Adjazenz Matrix, da sie ja während der Prozesses an Größe verliert und somit angepasst werden muss.

z2: Setzt den Laufindex 'index' auf Startposition 0

z3: Solange der Index 'index' echt kleiner als die Grenze 'grenze' ist:

```
  z4-6: Berechne die Zusammenhangskomponente des Knotens mit Knotennumm
er 'index' durch
        lokaler_zusammenhang. Nutze das Resultat zum die Adjazenz Matri
x und die Knotenwertliste durch
        Verbindungen_zusammenziehen um eine Zusammenhangskomponente zu
verkürzen. Fahre mit
        der neuen Adjazenz Liste und den neuen Knotenwerten fort.
  z7: Aktualisiere die Grenze, also setze sie gleich der neuen Zeilenlä
nge der Adjazenz Matrix.
  z8: Aktualisiere den Laufindex 'index'.
```

z9: Gebe die verkürtzte Adjazenz Matrix und Knotenwertliste zurück.

```
def Minimales_Gitter(adjazensmatrix,knotenwerte):
    grenze = len(adjazensmatrix)
    index = 0
    while(index < grenze):
        temp = Verbindungen_Zusammenziehen(adjazensmatrix,knotenwerte,lokaler_Zusammenhang(knotenwer
te,Adjazenzmartix_in_Liste(adjazensmatrix),index))
        knotenwerte = temp[1].copy()
        adjazensmatrix = temp[0].copy()
        grenze = len(adjazensmatrix)
        index = index + 1
    return adjazensmatrix, knotenwerte
```

2.3.3 Verwendung für die Approximationsanalyse

2.3.3.1 Definition:

Sei 0 ein (kontinuierliches) zweidimensionales Objekt, dass von einem diskreten Sensornetzwerk S mit Verbindungen A abgetastet wird. Die ***Homologie des Objektes O*** *wird von (S,A)* ***richtig erkannt****, falls kein Sensor s, der das bestehende Sensornetzwerk inclusive der Verbindungen* a_s *erweitert, gefunden werden kann, sodass der Minimalen Charakterisiergraphen von* (S, A) *und der Auswertung* $h_{(S,A)}$ *von* (S, A) *nicht schon dem Minimale Charakterisiergraph von* $(S \cup s, A \cup a_s), h_{(S \cup s, A \cup a_s)}$ *entspricht.* ***Die homologische Struktur*** *eines Objektes O ist somit der Minimale Charakterisiergraph einer richtigen Abtastung von O.*

Beweis:

Angenommen die Homologie von O wird nicht richtig erkannt, so existiert also ein Abschnitt von O, der noch nicht erfolgreich durch Sensoren aus (S,A) abgetastet wurde, also ein Bereich, in dem Sensoren nicht bereits einer in (S,A) vorhandene Zusammenhangskomponente zugeordnet werden können, oder Zusammenhangskomponenten existieren, die zusammenfallen müssen, da andernfalls die Annahme des nicht richtigen Erkennens der Homologie verfällt. Somit existieren also Sensoren inclusive neuer lokal in (S, A) *eingebettete Verbindungen, die* (S, A) *verändern, da ein Sensorbereich gefunden wurde, der (eine) neue Zusammenhangskomponenten bildet, oder Zusammenhangskomponenten zusammenzieht, also insbesondere das Minimale Gitter verändert.* ■

2.3.3.2 Proposition:

Zwei (verschieden) diskrete Sensornetzwerke (S_1, A_1) *und* (S_2, A_2) *über einem Objetk O mit diskreten Höhenfunktionen* h_1 *und* h_2 *erfassen die gleiche homologische Struktur von O, falls ihr Minimaler Charakterisiergraph übereinstimmt,* $M_{graph}((S_1, A_1))$ *entspricht also* $M_{graph}((S_2, A_2))$*. Übereinstimmung meint dabei, es existiere eine Permutationsmatrix P, sodass* $P * A_{M_{graph}((S_1,A_1))} = A_{M_{graph}((S_2,A_2))}$ und gleichzeitig auch $P * h\big(M_{graph}((S_1, A_1))\big) = h(M_{graph}((S_2, A_2)))$.

Beweis:

Die Aussage folgt direkt aus Definition 2.3.3.1. Die Art der Übereinstimmung folg aus der Tatsache, dass ein den Sensoren über eine Nummerierungsfunktion eine Nummer zugeordnet

wird, diese Zuweisung ist nicht eindeutig, siehe 2.2.2 die Funktion s_n. Somit existieren auch mehrere Darstellungen desselben Charakterisiergraphen. ■

Es müssen also alle Kombinationen unter Berücksichtigung der Beziehungserhaltung der Sensorwerten und Sensorverbindungen verglichen werden. Da die Sensorwerte sortiert werden können, und nicht zu viele Sensoren gleichen Knotenwertes innerhalb des Minimalen Charakterisiergraphen existieren, da sie sonst ja zusammengeführt werden würden, kann die Probe verbessert werden. Nur Sensoren gleichen Knotenwertes können zum Beispiel nicht in einem Minimalem Charakterisiergraphen enthalten sein, der zusammenhängende Verbindungen realisiert. Der folgende Algorithmus verkürzt den Vergleichsprozess anhand der Vorsortierung der Sensorwerte.

2.3.4 Vergleich „Minimaler Charakterisiergraphen“

Da es verschiedene Möglichkeiten gibt, denselben Charakterisiergraphen als Kombination des Wertefeldes und der Adjazenz Matrix darzustellen, muss der Vergleich formalisiert werden. Der folgende Abschnitt formalisiert den Vergleich von zwei Minimalen Charakterisiergraphen im nummerischen Sinne. Dabei werden zwei zu vergleichende Minimale Strukturen den Sensorwerten nach sortiert, die Beziehungen der Verbindungen eingeschlossen, und dann die Kombinationen überprüft, die Sensoren gleichen Sensorwertes noch beinhalten. Dazu wird die Adjazenz Matrix der unterschiedlichen Umsortierungen verglichen. Sollte keine Übereinstimmung gefunden werden, so stimmen die minimalen Graphen nicht überein.

Funktion: Transponiere_Knotenbenennung

In:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen (s.0.)
Knotenwerte; Liste mit Knotenwerten,
             Knoten i hat den Knotenwert: (Knotenwerte[i])
i; Integer, Index des Knotens, dessen Bennennung in j geändert werden soll.
j; Integer, Index des Knotens, dessen Bennennung in i geändert werden soll.
```

Out:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen, Zeile und Spalte i wurden mit der Zeile und
                 Spalet j vertauscht
Knotenwerte; Liste mit Knotenwerten,der Knotenwert i wurde mit dem Knotenwert j vertauscht
```

Kommentar:

z1-6: Vertausche die i-te Zeile der Adjazenz Matrix mit der j-ten Zeile, Vertausche den i-ten und j-ten Eintrag in Knotenwerte, jeweils durch Dreieckstausch z7: k durchlauft die Zeilen der Adjazensmatrix:

```
z8-10: Tausche in Zeile k die Einträge i und j, also insgesamt die Spalten i und j.
```

```
def Transponiere_Knotenbenennung(Adjazenz_Matrix, Knotenwerte,i,j):
    temp_x = Adjazenz_Matrix[i].copy()
    temp_k = Knotenwerte[i]
    Adjazenz_Matrix[i] = Adjazenz_Matrix[j].copy()
    Knotenwerte[i] = Knotenwerte[j]
    Adjazenz_Matrix[j] = temp_x.copy()
    Knotenwerte[j] = temp_k
    for k in range(0,len(Adjazenz_Matrix[0])):
        temp_y = Adjazenz_Matrix[k][i]
        Adjazenz_Matrix[k][i] = Adjazenz_Matrix[k][j]
        Adjazenz_Matrix[k][j] = temp_y
    return Adjazenz_Matrix, Knotenwerte
```

Funktion: Insertionsort_mit_Permutation

In:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen (s.o.)
Knotenwerte; Liste mit Knotenwerten,
             Knoten i hat den Knotenwert: (Knotenwerte[i])
```

Out:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen, Die Matrixeinträge sind nun nach der
                 Umordnung der zugehörigen Knotenwerte angeorden, die nach ihren Werten sortiert sind
Knotenwerte; Liste mit Knotenwerten, die Knotenwerte sind nach ihren Werten sortiert
```

Kommentar:

z1-7: Regulärer Insertionsort, indem parallel die Matrixeinträge mit getauscht werden.

```
def Insertionsort_mit_Permutation(Adjazenz_Matrix, Knotenwerte):
    for i in range(1,len(Knotenwerte)):
        temp = Knotenwerte[i]
        j = i
        while( j > 0 and Knotenwerte[j-1] > temp):
            Adjazenz_Matrix = Transponiere_Knotenbenennung(Adjazenz_Matrix, Knotenwerte,j,j - 1)[0]
            j = j - 1
    return Adjazenz_Matrix, Knotenwerte
```

Funktion: Testpositionen_Vergleich_Min_Git

In:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen (s.o.), eines Minimalen
                 Charakterisiergraphen
Knotenwerte_geordnet; Liste mit Knotenwerten,
                      Knoten i hat den Knotenwert: (Knotenwerte[i]), die liste ist bei übergabe nach
                      den Werten sortiert
```

Out:

```
positionen; 2 dimesionale Liste, die Knotenpositionen von Knoten gleichen Knotenwertes vermerkt. Jedes
Listenelement enthält eine Liste mit Nummern von Knoten, die den gleichen Knotenwert haben. Es sind nur
Knotennummern enthalten, wenn es auch mehr als einen Knoten mit selben wert gibt.
```

Kommentar:

z1: Legt die leere Liste positionen an. z2: Setzt den Laufindex i auf 0. z3: Solange i nicht die Liste Knotenwerte_geordnet durchlaufen hat:

```
z4: Wenn i + 1 noch nicht den Listenindex von Knotenwerte_geordnet überschreitet sowie i und i+1 die
    selben Knotenwerte haben:

    z5: Lege eine temporäre Liste mit den einträgen i und i + 1 an, denn die Knotenwerte der Knoten haben
        gleiche Werte.
    z6: Setze einen Counter auf 2, da i und i+1 schon betrachtet wurden
    z7: Solange der Nachfolger noch den Gleichen Knotenwert hat:

        z8-9: Füge den Index zu temp hinzu, erhöhe den Counter.

    z10-11: Erweitere positionen um temp, den errechneten  Indexen der gleichwertigen Knoten, erhöhe i
            um den counter, da bis i + counter alles bearbeitet wurde.
z12: Andernfalls(zu z4): erhöhe i um 1, denn der Knotenwert von i und i+1 ist verschieden.
```

```
def Testpositionen_Vergleich_Min_Git(Adjazenz_Matrix, Knotenwerte_geordnet):
    positionen = []
    i = 0
    while(i <= len(Knotenwerte_geordnet)):
        if(i + 1 < len(Knotenwerte_geordnet) and Knotenwerte_geordnet[i] == Kn
otenwerte_geordnet[i + 1]):
            temp = [i] + [i + 1]
            count = 2
            while(i + count < len(Knotenwerte_geordnet) and Knotenwerte_geordn
et[i + count] == Knotenwerte_geordnet[i]):
                temp = temp + [i + count]
                count = count + 1
            positionen = positionen + [temp]
            i = i + count
        else:
            i = i + 1
    return positionen
```

Funktion: Permutiere_Min_Git

In:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen (s.o.), eines Minimalen
                 Charakterisiergraphen
Knotenwerte_geordnet; Liste mit Knotenwerten,
                      Knoten i hat den Knotenwert: (Knotenwerte[i]), die liste ist bei übergabe nach
                      den Werten sortiert
Knotennummern_neu_geordnet; Liste mit Knotenwerten, dieser Reienfolge nach sollen die Knoten und ihre
                            Verbindungen umsortiert werden
```

Out:

```
Adjazenz_Matrix; 2 dimesionale Liste der Knotenverbindungen, Die Matrixeinträge sind nun nach der
                 übergebenden Umordnung getauscht
Knotenwerte; Liste mit Knotenwerten, die Knotenwerte sind eben so angeordnet.
```

Kommentar:

z1: Legt temporäre Variabele der Kopien der Adjazenz_Matrix und der Knotenwerte_geordnet an z2: Für i von 0 bis zur Anzahl der Knoten - 1:

```
    z3: Falls der i-te Eintrag in Knotenwerte_neu_geordnet echt größer i selbst ist:

        z4: setze temp gleich dem vorherigen temp mit Zeilen und Spalten i und Knotenwerte_neu_geordnet[i]
            wie in Transponiere_Knotenbenennung beschrieben vertauscht.
```

```
def Permutiere_Min_Git(Adjazenz_Matrix, Knotenwerte_geordnet, Knotennummern_neu_geordnet):
    temp = [Adjazenz_Matrix.copy(), Knotenwerte_geordnet.copy()]
    for i in range(0,len(Knotenwerte_geordnet)):
        if(Knotennummern_neu_geordnet[i] > i):
            temp = Transponiere_Knotenbenennung(temp[0].copy(), temp[1].copy(),i,Knotennummern_neu_geordnet[i])
    return temp
```

Der folgende Algorithmus ist die nicht rekursive Form des „Heap's algorithm“ von B. R. Heap (Heap, 1963):

Funktion: permutiere

In:

```
len_Sortierbereich, Integer; Länge des zu betrachteten Zahlenbereiches.
zu_sortierende_Werte,  Liste von Integern; Listen von Knotennummern, die
                                          (im Funktionszusammenhang) gleiche Knotenwerte aufweisen.
```

Out:

```
merke_Werte; 2 dimensionale Liste; jedes Listenelement entspricht einer der Möglichen Permutationen der
             Zahlen in zu_sortierende_Werte auf eine Liste der Länge len_Sortierbereich
```

Kommentar:

z1-23: Der Folgende Programmtext entspricht der nicht-recrusiven Form des 'Heap's algorithm'. Dieser Algoritmus Berechnet alle Anordnungen verschiedener Zahlen aus zu_sortierende_Werte auf eine Liste der Länge len_Sortierbereich.

```
def permutiere(len_Sortierbereich, zu_sortierende_Werte):
    merke_Werte = []
    status = []
    for i in range(0,len_Sortierbereich):
        status = status + [0]
    merke_Werte = merke_Werte + [zu_sortierende_Werte.copy()]
    i = 0
    while (i < len_Sortierbereich):
        if (status[i] < i):
            if (i%2 == 0):
                temp = zu_sortierende_Werte[0]
                zu_sortierende_Werte[0] = zu_sortierende_Werte[i]
                zu_sortierende_Werte[i] = temp
            else:
                temp = zu_sortierende_Werte[status[i]]
                zu_sortierende_Werte[status[i]] = zu_sortierende_Werte[i]
                zu_sortierende_Werte[i] = temp
            merke_Werte = merke_Werte + [zu_sortierende_Werte.copy()]
            status[i] = status[i] + 1
            i = 0
        else:
            status[i] = 0
            i = i + 1
    return merke_Werte
```

Funktion: Testmenge_Auswertungsfolge

In:

```
n, Integer; (Im Funktionszusammenhang) die länge des Knotenwertfeldes.
tauschpositionen, 2 dimensionale Liste; Liste aller Indexe von Knoten mit gleichen Knotenwerten, dabei
                                        sind Knoten des Gleichen Knotenwertes zusammengefasst. (s.0)
```

Out:

```
Folgenliste, 2 dimensionale Liste; Liste der verschiedenen Anordnungen der Zahlen 0 bis n-1, in denen nur
Zahlen aus der tauschpositionen Liste jeweils nur mit Zahlen aus dem selben Listeneintrag vertaucht werden
```

Kommentar:

z1: Legt eine leere Variabele als Vorlage der Liste an

z2/3: Erstellt eine Liste der Zahlen 0 bin n-1

z4: Legt eine leere Variabele als Speicheer der Wiederholungsrate der einzelnen Kombinationsbausteine an. (s.u.)

z5-8: Legt eine Liste der Anzahlen der Möglichkeiten an, die im i ten Schritt schon durch die Vertauschungen von tauschpositionen(1 bis i) entstehen. z9: Legt eine leere Variabele an, in der die Möglichkeiten gespeichert werden sollen.

z10: Setzt die Variabele, mit der die Gesamtzahl der Möglchhkeiten berechnet werden soll auf eins

z11/12: Berechnet die Anzahl der Möglichkeiten über die Liste 'Kombinationen'

z13/14: Füllt das Ausgabefeld mit der Forlagefolge, so oft wie es Möglichkeiten gibt

z15-21: Fügt in passender Reihenfolge die einzeln berechneten Teillisten, die Indizes gleicher Knoterwerte enthalten, zu einer möglichen Umordnung zusammen. Dabei berechnet die Funktion Tausch_Positionen die Teilvertauschungen, die dann zusammengefügt werden. Passend zusammenfügen geschieht auf folgende Weise: Hat der erste Eintrag in Tauschpositionen r Möglichkeiten, und der zweite v, so entstehen für jede der v Möglichkeiten r neue, also insgesamt (r * v) ,... Die dem ersten Eintrag in Tauschpositionen zugehörigen Kombinationen wechseln also Spaltenweise, die der zweiten erst jede r -te Spalte, usw..

```
def Testmenge_Auswertungsfolge(n, tauschpositionen):
    Folgenvorlage = []
    for i in range(0,n):
            Folgenvorlage = Folgenvorlage + [i]
    Kombinationen = []
    Positionen = [len(tauschpositionen[0])]
    for j in range(0,len(tauschpositionen)):
        Kombinationen = Kombinationen + [permutiere(len(tauschpositionen[j]),tauschpositionen[j])]
        if(j!=0):
            Positionen = Positionen + [len(tauschpositionen[j]) * Positionen[j-1]]
    Folgenliste = []
    anzahl_gesamt_Kombinationen = 1
    for k in range(0,len(Kombinationen)):
        anzahl_gesamt_Kombinationen = anzahl_gesamt_Kombinationen * len(Kombinationen[k])
    for h in range (0, anzahl_gesamt_Kombinationen):
        Folgenliste = Folgenliste + [Folgenvorlage]
    for l in range(0,anzahl_gesamt_Kombinationen):
        for m in range(0, len(tauschpositionen)):
            for o in range(0,len(Kombinationen[m])):
                if(m == 0 and l% len(tauschpositionen[m]) == o):
                    Folgenliste[l] = Tausch_Positionen(Folgenliste[l],[Kombinationen[m][o]]).copy()
                if (m != 0 and  int(l/ (Positionen[m-1])) % len(Kombinationen[m]) == o ):
                    Folgenliste[l] = Tausch_Positionen(Folgenliste[l],[Kombinationen[m][o]]).copy()
    return Folgenliste
```

Funktion: Tausch_Positionen

In:

```
folge,  Liste von Integern von 0 bin n - 1
tauschpositionen: zweidimensionale Liste von Integern, Liste mit Liste in folge zu tauschender Positionen
```

Out:

```
folge; Nach der Reihenfolge der übergebenen tauschpositionen geänderte folge
```

Erklärung:

Permutiert die Liste folge der Liste tauschpositionen nach. Die Eingabe ([0,1,2,3],[[2,1]]) würde Beispielsweise [0,2,1,3] bewirken. Diese Funktion existiert nur, um die Funktion Testmenge_Auswertungsfolge übersichtlicher und verständlicher zu gestalten.

```
def Tausch_Positionen(folge,tauschpositionen):
    position = 0
    position_in_tauschpositionen = 0
    for i in range(0,len(folge)):
        for j in range(0,len(tauschpositionen[position])):
            if(i == tauschpositionen[position][j]):
                folge[i] = tauschpositionen[position][position_in_tauschpositionen]
                position_in_tauschpositionen = position_in_tauschpositionen + 1
        if(position_in_tauschpositionen == len(tauschpositionen[position]) and position < len(tauschpositionen) - 1):
            position_in_tauschpositionen = 0
            position = position + 1
    return folge
```

Funktion: Vergleiche_Minimales_Gitter

In:

```
Verbindungen_a, Adjazenz Matrix des ersten übergebenen Minimalen Gitters.
Knotenwerte_a, Knotenwerte des ersten übergebenen Minimalen Gitters.
Verbindungen_b, Adjazenz Matrix des zweiten übergebenen Minimalen Gitters.
Knotenwerte_b, Knotenwerte des zweiten übergebenen Minimalen Gitters.
```

Out:

```
(ist_gleich/ist_nicht_gleich) Kommentar, Ergebniss der Überprüfung der beiden Charakterisiergraphen auf
Übereinstimmung im Sinne der Proposition 2.3.3.2.
```

Kommentar:

z1/2: Kommentare als Variabeln setzen.

z3/4: Falls die Länge der Knotenwerte a und b nicht übereinstimmen, so können sie nicht gleich sein. Gebe den entsprechenden Kommentar zurück

z5/6: Speichere die durch Insertionsort_mit_Permutation sortieren Minimalen Charakterisiergraphen in einer Themporären Variabele.

z7/9: Vergleiche die Knotenwerte von Temp_a, Temp_b der Reihe nach. Da eine Sortierung vorliegt, müssen die Werte für jeden Index i den Gleichen Wert annehmen.

z10: Da Knotenwerte mehrfach Vorkommen können, ist eine Sortierung der Werte nicht ausreichen um die Minimalen Charakterisiergraphen zu vergleichen, so müssen die verschiedenen Anordnunegn der Knotenwerte zusammen mit den Verbindungen, die den gleichen Knotenwert haben, getestet werden. Dazu wird eine Menge der Möglichen Anordnungen erzeugt und temporär gespeichert.

z11-14: Vergleicht die nach den abgespeicherten Möglichkeiten umsortierten Graphen von b mit dem sortieren festgehaltenen Graphen von a.

```
def Vergleiche_Minimales_Gitter(Verbindungen_a,Knotenwerte_a,Verbindungen_b,Knotenwerte_b):
    ist_gleich = "Die Minimalen Charakterisiergraphen stimmen überein!"
    ist_nicht_gleich = "Die Minimalen Charakterisiergraphen stimmen nicht überein!"
    if(len(Knotenwerte_a) != len(Knotenwerte_b)):
        return ist_nicht_gleich
    Temp_a = Insertionsort_mit_Permutation(Verbindungen_a,Knotenwerte_a)
    Temp_b = Insertionsort_mit_Permutation(Verbindungen_b,Knotenwerte_b)
    for i in range(0,len(Knotenwerte_a)):
        if(Temp_a[1][i] != Temp_b[1][i]):
            return ist_nicht_gleich
    Kombinationen_b = Testmenge_Auswertungsfolge(len(Knotenwerte_b),Testpositionen_Vergleich_Min_Git
(Temp_a[0],Temp_a[1]))
    for j in range(0,len(Kombinationen_b)):
        if(Temp_a[0] == Permutiere_Min_Git(Temp_b[0],Temp_b[1],Kombinationen_b[j])[0]):
            return ist_gleich
    return ist_nicht_gleich
```

2.4 Inhaltsverzeichnis des Programmteiles

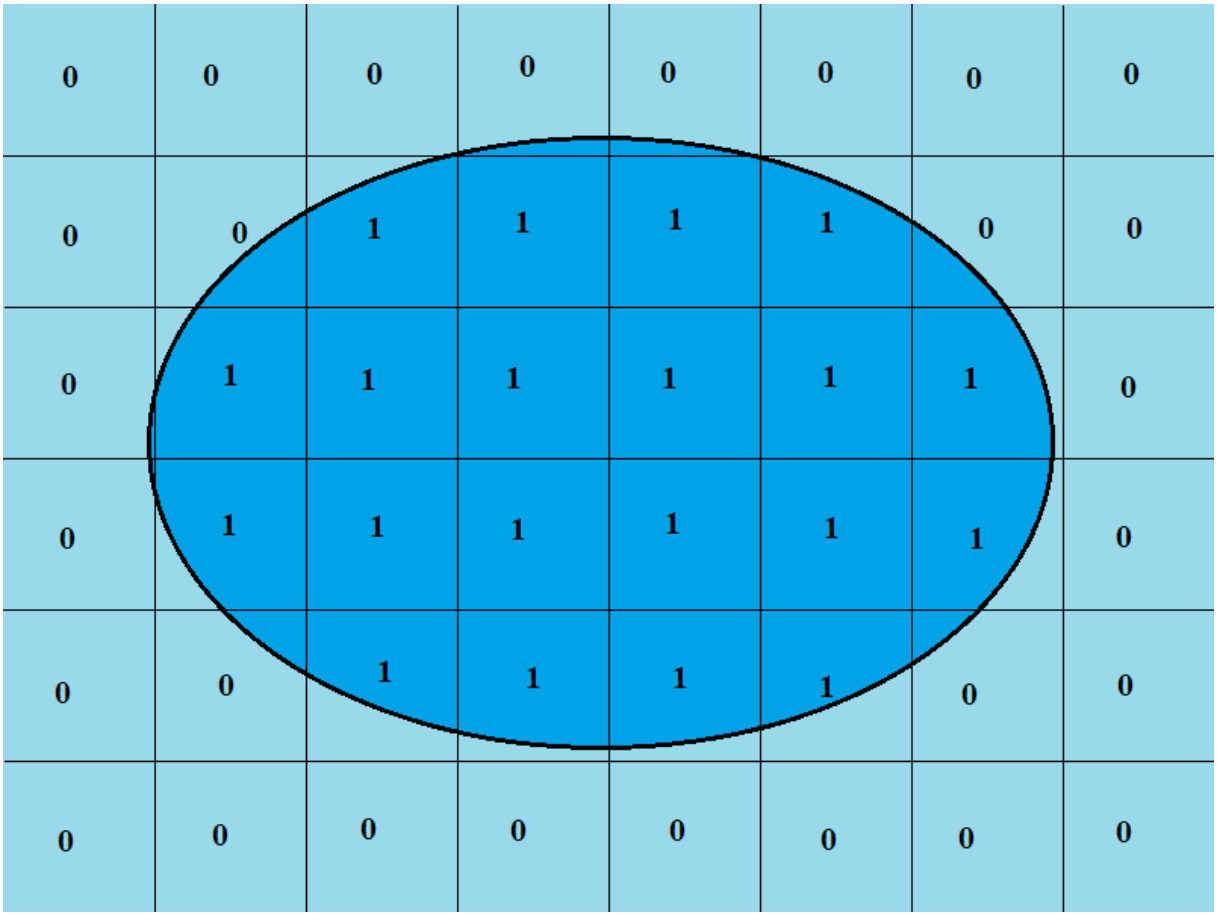

Abbildung 8: Darstellung einer Ellipse in Form einer nxm Matrix. Die Einträge der Matrix entsprechen der Anzahl an überliegenden Objekten. Dabei geht Genauigkeit verloren.

Da ein Programm keine kontinuierlichen Testgebiete darstellen kann, muss auf diskrete Darstellung zurückgegriffen werden. Diese diskrete Darstellung ist oft eine Art geometrische Approximation eines kontinuierlichen Testgebietes. Dabei wird die Höhenfunktion des Gebietes approximiert. Ein Kreis kann beispielsweise nicht durch endlich viele „Pixel“ dargestellt werden. „Pixel“ meinen hier Bildquadrate, also Quadrate, in denen die Höhenfunktion einen konstanten wert annimmt, ähnlich der Pixel eines Bildschirmes, die pro Zeitintervall nur eine Farbe anzeigen können. Die im weiteren Verlauf oft „Bild“ genannten diskreten Testgebiete sind somit geometrische Approximationen der darzustellenden Objekte. Die resultierenden Probleme dieses Ansatzes werden in 2.6.2 behandelt. Nach dieser Darstellungsweise lässt sich die Höhenfunktion eines rechteckigen Gebietes dann als Matrix darstellen, in dem dieses Gebiet von einem quadratischen, gleichmäßigen Koordinatennetz überzogen wird, wie in Abbildung 8 zu sehen ist. Der Eintrag der Matrix in (i, j) entspricht dann dem approximierten Wert der Höhenfunktion in der Netzzelle (i, j) des Koordinatennetzes.

Somit erhält man die diskrete Darstellung einer Höhen Funktion eines Gebiets. Darauf kann ein Sensornetzwerk aufgebaut werden. Sensoren können somit eine Zelle des Bildes abtasten und geben ihren Wert zurück. Dabei müssen nicht alle Zellen von Sensoren ausgewertet werden, und somit kann die die diskrete Charakteristik dieser (geometrischen) Approximation durch verschieden Sensornetzwerke ausgewertet werden. Dabei sind auch Netzwerke interessant, die viel weniger Sensoren als Netzwerkzellen nutzen, und dennoch die Struktur des Objektes gut auszuwerten zu scheinen. Zudem können nicht nur verschiedene Sensorpositionen, sondern auch verschiedene Sensorverbindungen getestet werden. Ein Sensornetzwerk über einem Bild ist beinhaltet also die Koordinaten der Sensoren, und die Verbindungen der Sensoren untereinander. Zusammen mit dem Bild ergibt sich so eine diskrete Höhenfunktion, die das Sensornetzwerk ausliest. Zusammen mit den Sensorverbindungen kann dann aus der Höhenfunktion und der Verbindung der Sensoren, die die Höhenfunktion ausgeben, die Summe aus *Theorem 4.3* (Yury Baryshnikov & Ghrist, 2009) berechnet werden. Die Verbindungen werden dabei in Form von Adjazenz Matrizen oder Adjazenz Listen dargestellt. Im folgenden Teil werden auch Programme zur Erstellung von Gittern und Bildern wie beschrieben wurde vorgestellt, sowie Testprogramm, die dem Verständnis oder der Handhabung dienen. Im Anhang wird die Verwendete Struktur und der Aufbau nochmal anhand des Programmes dargestellt. Folgend das Inhaltsverzeichnis des Programmcodes.

1. Berechnung der Formel aus Theorem 4.3 via Adjazenz Matrix

- euler_calcus_2d
- beta_0
- verbindungssuche_echtgr
- verbindungssuche_kl

Funktionen der nummerischen Berechnung von *Theorem 4.3* über die Adjazenz Matrix, siehe dazu Abschnitt 2.2.3.

2. Berechnung der Formel aus Theorem 4.3 via Adjazenz Listen

- Adjazenzmartix_in_Liste
- Adjazenzliste_in_Martix
- euler_calcus_2d_ad
- beta_0_ad

- verbindungssuche_echtgr_ad
- verbindungssuche_kl_ad

Funktionen der nummerischen Berechnung von *Theorem 4.3* über die Adjazenz Liste, siehe dazu Abschnitt 2.2.4.

3. Berechnung des "Minimalen Charakterisiergraphen"

- lokaler_Zusammenhang
- lokaler_Zusammenhang_recrusiv
- Verbindungen_Zusammenziehen
- kenn_verknüpfung
- Minimales_Gitter

Die Funktionen zu Abschnitt 2.3, siehe 2.3.2 für die genaue Implementierung und 2.3.1 für die Idee.

4. Approximationsbetrachtung

- Anzahl_gleichwertiger_und_verschiedenwertiger_Nachbarknoten

Die Funktion berechnet dem Namen nach die Anzahl der gleichwertigen, verschiedenen verschiedenwertigen und verschiedenwertigen Nachbarn, siehe 2.7.2.

- Transponiere_Knotenbenennung
- Insertionsort_mit_Permutation
- Testpositionen_Vergleich_Min_Git
- Permutiere_Min_Git
- permutiere
- Testmenge_Auswertungsfolge
- Tausch_Positionen
- Vergleiche_Minimales_Gitter

Funktionen zum vergleichen zweier Minimaler Charakterisiergraphen. Nach Definition 2.3.3.1 gibt es verschiedene Zuordnungsfunktionen, und somit verschiedene Darstellungen des gleichen Minimalen Gitters. Die Funktion vergleicht somit die Gitter unter diesen Gesichtspunkten, siehe 2.3.4.

5. Berechnungshilfen zur weiteren Untersuchung

- berechne_Knotenwerte

Die Funktion berechne_Knotenwerte liefert die Sensorwerte eines Sensornetzwerkes, dessen Sensorkoordinaten übergeben wurden, die Sensorwerte der Sensoren entsprechen dabei den Werten, die Sensoren an den übergebenden Positionen des übergebenden diskreten Gebietes annehmen hätten

- berechnung_approx_euler

Diese Funktion berechnet zu einem übergebenen diskreten Gebiet und diskreten Sensornetzwerk die aus diesen Daten resultierende Summe des *Theorems 4.3* über Adjazenz Matrizen realisiert. Dazu werden die Funktionen berechne_Knotenwerte, und euler_calcus_2d aufgerufen.

- berechnung_approx_euler_ad

Diese Funktion berechnet zu einem übergebenen diskreten Gebiet und diskreten Sensornetzwerk die aus diesen Daten resultierende Summe des *Theorems 4.3* über Adjazenz Listen realisiert. Dazu werden die Funktionen berechne_Knotenwerte, und euler_calcus_2d_ad aufgerufen.

- berechnung_Minimaler_Charakterisierbaum

Auch diese Funktion berechnet zu einem übergebenen diskreten Gebiet und diskreten Sensornetzwerk den resultierenden Minimalen Charakterisiergraphen. Dazu wird die Funktion Minimales_Gitter aufgerufen.

6. Gitter und Gitteroperationen

- ist_symetrich

Diese Funktion überprüft eine Adjazenz Liste auf Beziehungen, die nicht symmetrisch sind, dass heißt auf Sensorbeziehungen, die einseitig sind. Da sich Sensoren aber gegenseitig kennen liegt somit ein Fehler in der Verbindung Liste vor. Da diese sehr übersichtlich werden, vor allem wenn große Gitter in einzelnen Punkten verändert werden sollen, ist diese Funktion eine gute Kontrolle.

- highlight

Diese Funktion hebt Koordinaten eines Gebietes optisch hervor.

Auch dies dient als Hilfsmittel der Approximationsbetrachtung. Werden die Daten der Sensoren nach der Funktion anzahl_gleichwertiger_und_verschiedenwertiger_Nachbarn aufgelistet, kann somit die Position des Sensors mit bestimmten Werten hervorgehoben werden.

- Gitter_Scalierend

Gitter_Scalierend gibt zu einem übergebenden Bild

- Gitter_linear_Scalierend
- Gitter_linear_Schachbrett_Scalierend

Gitter analog setzen:

- Knoten_hinzufügen

Fügt einen Knoten hinzu, der Vorerst noch nicht in den Verbindungen enthalten ist.

- Neues_Gitter

Erstellt ein neues Gitter, in dem die Sensoren nur sich selbst kennen.

- Verbindung_neuer_Knoten

Erweitert die Verbindungen um die neuen Knoten, die vorerst nur sich selbst kennen.

- Verbindung_setzen
- Verbindungen_setzen

Diese Funktionen setzen die Verbindung von einem Knoten mit einem bzw. mehrerer anderer Knoten auf verbunden.

- Verbindung_zurücksetzen

Diese Funktion setzt die Verbindung der zwei übergebenden Knoten zurück.

- Knoten_Loeschen

Diese Funktion entfernt einen Knoten samt Verbindungen.

7. Bild und Bildoperationen

- leeres_Bild

Erstellt eine $n\,x\,m$ Matrix mit allen Einträgen Null. Darauf können dann Objekte durch die Höhenfunktion dargestellt werden.

- Viereck
- in_Viereck

Diese Funktion fügt unter gewissen Voraussetzungen der Eingabewerte ein Viereck als Höhenfunktion diskret in einem Bild ein. Die Voraussetzungen sind im Programmteil beschrieben. Diese Funktion sollte immer überprüft werden, ob sie auch das gewünschte Ergebnis erzielt. Dies kann mit der Funktion *Bildabschnitt*, die weiter unten beschrieben wird geschehen, mit der die Bilder in Abschnitten Betrachten und kontrolliert werden können. Dies gilt für alle Bildgenerierenden Funktionen.

- Viereck_inv
- Viereck_mit_Loch

Diese Funktion fügt unter gewissen Voraussetzungen der Eingabewerte ein Viereck, welches ein ebenfalls viereckiges Loch enthält, als Höhenfunktion diskret in einem Bild ein. Es gelten die gleichen Voraussetzungen wie bei der *Viereck* Funktion.

- Kreis
- Torus

Diese Funktionen fügen einen Kreis, bzw. einen Ring als Höhenebenen in ein Bild ein. Auch hier sollten die Eingabebedingungen beachtet werden.

- Punkt
- Punktmenge

Hiermit können Abstrakte Gebilde als Bild realisiert werden, indem die Koordinaten der Höhenebenen des Gebildes übergeben werden.

- Bildabschnitt

Zeigt den Abschnitt eines Bildes, um bei sehr großen Bildern die Bild-Generierenden Funktionen zu überprüfen, und die Hervorhebungen der highlight Funktion genauer zu betrachten.

2.5 Untersuchung der approximativen Eigenschaft der Realisierung

2.5.1 Vorgehensweise der Untersuchung

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 0],
 [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Abbildung 9: Die diskrete Zählfunktion von fünf Zielträgern in Form von Rechtecken auf einem quadratischen Gebiet.

Abbildung 9 zeigt eine wie Abbildung 8 eine diskrete Form der Darstellung der Höhenfunktion von fünf Rechtecken in einem rechteckigen Gebiet. Dabei werden Zielträger im Folgenden oft als Objekte behandelt, um auf die verschiedenen Formen der Zielträger eingehen zu können. Auf diesem Bild wird nun ein Gitter erstellt, das in folgender Berechnung erstmal jeden Bildpunkt einem Sensor zuteilt. Diese Sensoren sind geometrisch lokal verbunden, Sensoren kennen ihre (geometrischen) Nachbarn. Somit kann auf dieser Grundlage mit den beschriebenen Funktionen aus 2.2.3 und 2.2.4 auf dieser Basis das Euler Integral nach *Theorem 4.3* berechnet werden:

```
Grundlagen_Gitter = Gitter_Scalierend(Grundlage_0, 1)
```

```
berechnung_approx_euler_ad(Bild_0 ,Grundlagen_Gitter)
```

```
5
```

Da ein Rechteck die Euler Charakteristik eins aufweist, ergibt diese Berechnung das korrekte Ergebnis 5. Es kann außerdem der Minimale Charakterisiergraph gebildet werden. Die Matrix entspricht der Verbindungsmatrix (Adjazenz Matrix) der Sensoren mit Werten in der darunter

dargestellten Liste. Sensor i hat dabei den i-ten Listeneintrag als Sensorwert, die i-te Zeile und Spalte als Verbindungen. Aus der programmtechnischer Sicht beginnt die Sensornummerierung bei 0.

```
berechnung_Minimaler_Charakterisierbaum(Bild_0 ,Grundlagen_Gitter)
```

```
([[1, 1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 1],
  [1, 1, 1, 0, 0, 0],
  [1, 1, 0, 1, 0, 0],
  [1, 1, 0, 0, 1, 0],
  [1, 1, 0, 0, 0, 1]],
 [0, 1, 2, 2, 2, 2])
```

Anhand dieser Berechnung kann nun die Approximation mit der anderer oder anders Skalierter Sensornetzwerke verglichen werden.

```
Grundlagen_lineares_Gitter = Gitter_linear_Scalierend(Grundlage_0, 2)
```

```
berechnung_approx_euler_ad(Bild_0 ,Grundlagen_lineares_Gitter)
```

```
3
```

```
berechnung_Minimaler_Charakterisierbaum(Bild_0 ,Grundlagen_lineares_Gitter)
```

```
([[1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0],
  [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1],
  [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1],
  [0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0],
  [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1],
  [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1]],
 [0, 1, 1, 2, 1, 2, 0, 1, 2, 2, 1, 0])
```

Diesmal wird nur jeder Bildpunkt betrachtet, dessen x und y Koordinaten jeweils Modulo zwei den Wert 0 ergeben. Also die Kreuzpunkte jeder zweiten Zeilen und Spalten. Die Sensoren sind wieder lokal verbunden. Diesmal werden mehr Zusammenhangkomponenten erkannt. Ungenauigkeit führt nicht nur zu weniger Komponenten, die auf Grund ungenauer Abtastung nicht erkannt werden. Komponenten, die eigentlich zusammengehören, wurden nicht als zusammenhängend erkannt. Dies liegt auch an den (lokalen) Verbindungen der Sensoren. Liegen zwei Sensoren innerhalb einer Zusammenhangskomponente, sind aber nicht direkt verbunden, so ergibt sich eine neue Zusammenhangskomponente die eigentlich nicht existiert. Somit lesen vor allem lokale Realisierungen der Verbindungen vielleicht oft mehr Komponenten als eigentlich existieren. Die folgende Grafik soll dies veranschaulichen.

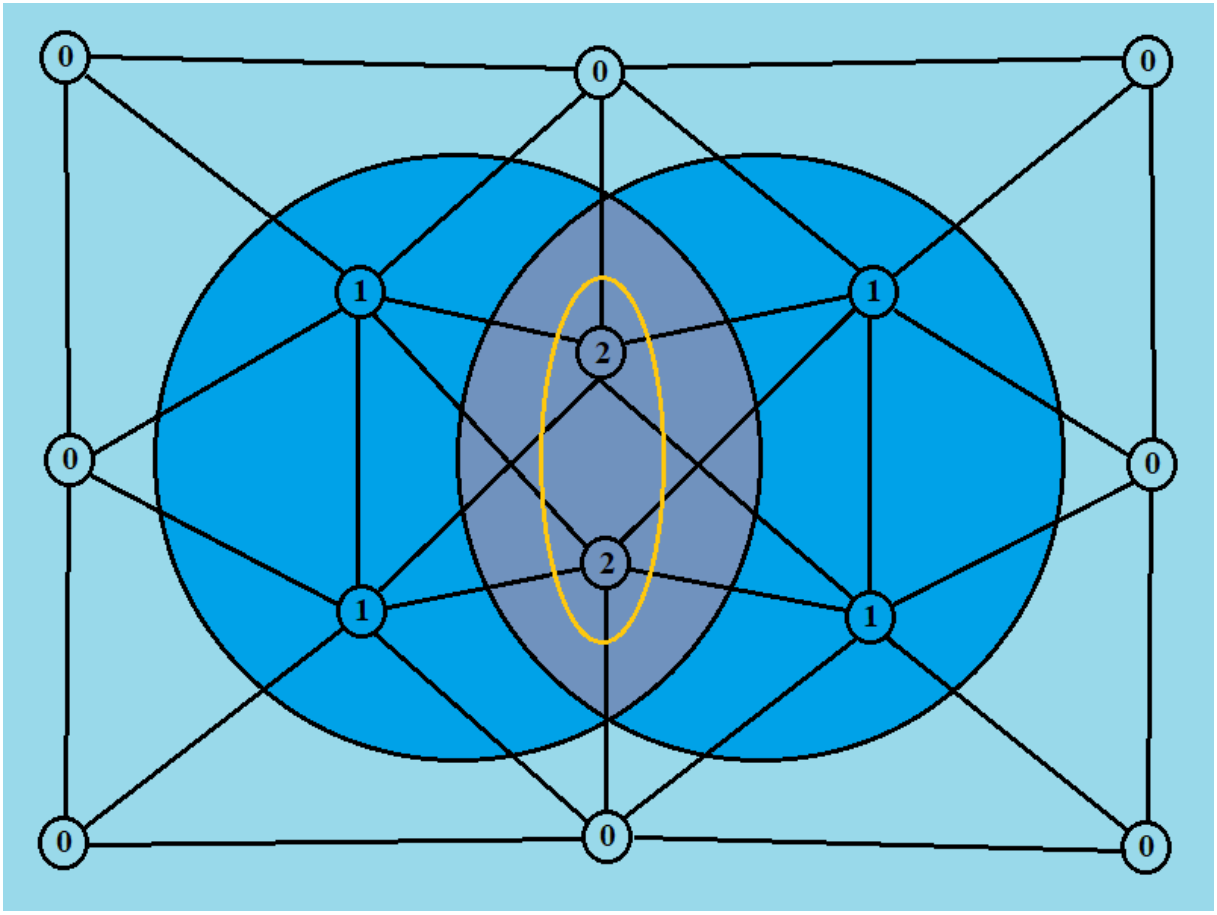

Abbildung 10: Die diskrete Abtastung des Gebietes mit zwei überlappenden Kreisen enthält eine direkte Verbindung zweier Sensoren innerhalb der grauen inneren Zusammenhangskomponente nicht. Somit erkennt das Sensornetzwerk die graue Zusammenhangskomponente nicht als eine, sondern als 2.

In der Abbildung 10 sind die eingekreisten Sensoren im grauen Bereich nicht direkt verbunden. Somit erkennt der Algorithmus zwei Komponenten an Stelle einer. Somit kann anhand der Minimalen Charakterisiergraphen den Problemen der verwendeten Gitter auf den Grund gegangen werden. Es werden verschiedene Formen der Verbindungen, verschieden Skalierte lineare Gitter und verschiedene Bildgenauigkeiten getestet werden.

Eine weitere Möglichkeit der genaueren Betrachtung verschiedener Gitterauswertungen bietet die in Abschnitt 2.7.1 vorgestellte Funktion, die die für jeden Sensor die Anzahl gleichwertiger und verschiedenwertiger, direkt verbundener Sensoren zählt, und diese jeweils abspeichert. Summiert man diese jeweils für alle Sensoren auf, und teilt sie durch die Anzahl der Sensoren des Netzwerkes, so erhält man den Durchschnitt direktverbundener gleichwertiger und verschiedenwertiger Sensornachbarn. Das Verhältnis dieser beiden Werte berechnet dann folgende Funktion, also

$(Durchschnitt\ gleichwertiger\ Nachbarn)/$
$(Durchschnitt\ anderswertiger\ Nachbarn).$

Dies ergibt mit dem Gitter aus der Berechnung mit resultierendem Wert 5 den Wert:

```
Berechne_Nachbarverhältnis(Bild_0 ,Grundlagen_Gitter)
```

```
2.3990825688073394
```

Für die zweite Berechnung, die das Ergebnis 3 lieferte, ergibt sich folgender Wert:

```
Berechne_Nachbarverhältnis(Bild_0 ,Grundlagen_lineares_Gitter)
```

```
1.8124999999999998
```

Die Vermutung hinter diesem Wert ist dabei folgende, je besser die (dargestellte) Objektkomplexität zur Sensoranzahl passt, und je besser das zugrundeliegende Sensornetzwerk an die Homologie des Objektes angepasst ist, desto höher wird der Wert. Die genaue Beschreibung dieser Begriffe und die Vermutung selbst soll unter anderem folgend betrachtet werden.

Vorerst gilt es aber die Verwendeten Teststrukturen darzustellen. Es werden viel verschiedene Bilder getestet, die jeweils in unterschiedlichen Bildgrößen dargestellt sind. Bild eins enthält dabei fünf Rechtecke, Bild zwei enthält vier Kreise, Bild drei enthält fünf Ringe und Bild 4 zwei Vierecke mit viereckigem Loch. Somit werden auch Objekte verschiedener Euler Charakteristik betrachtet, ebenso können Kreise und Ringe geometrisch nicht sehr gut durch Pixel dargestellt werden, im Gegensatz zu gut gewählten Rechtecken.

Jedes Bild wird in den Größen [50x50,100x100,500x500] und teilweise in [1000x1000] generiert, also durch Matritzen der entsprechenden Größe dargestellt. Dies und alle Berechnungen aus 2.5 sind unter Punkt 8 des Programmteiles zu finden.[…x]

Es werden hauptsächlich drei Sensornetzwerke betrachtet, die alle in Verschiedenen Skalierungen auf ein Bild angewandt werden können. Der Skalierungsfaktor beschreibt Feinheit des Netzwerkes in Folgender Form. Wird der Wert i gewählt, so werden Sensoren in allen Bildpunkten betrachtet, dessen x und y Koordinaten Modulo i den Wert Null ergeben. Somit bildet ein kleiner Wert für i ein sehr genau auswertendes Netzwerk, ein großer Wert ein entsprechend grobes Netzwerk. Das erste Sensornetzwerk, lokal genannt, realisiert lokale Verbindungen, jeder Sensor ist mit dem Sensor geometrisch direkt über ihm, direkt unter ihm,

und direkt neben ihm verbunden. Das zweite Sensornetzwerk, genannt senk_waagerecht, verbindet alle Sensoren, die entweder die gleichen x oder die gleichen y Koordinaten haben. Das dritte Netzwerk realisiert alle direkten Verbindungen, jeder Sensor ist mit jedem Sensor direkt verbunden. Alle drei Sensornetzwerke liefern für dasselbe Bild und denselben Skalierungsfaktor die gleichen Sensorwerte, da die gleichen Sensorkoordinaten genutzt werden, somit ändern sich nur die Verbindungen. Dies vereinfacht den Vergleich. Die Berechnungen des Euler Integrales werden alle samt mit der Realisierung über Adjazenz Listen berechnet, da die Bildmatrix und die Verbindungslisten sehr groß werden können. Dies ist auch der Grund warum im weiteren Verlauf einige Berechnungen fehlen, da die Rechenleistung bzw. die maximale Rekursionstiefe erreicht wurde.

Zu beachten ist noch, dass die Realisierung einen ganzzahligen Integralbegriff beinhaltet. Konvergenz ist somit begrenzt Betracht bar. Der maximal mögliche Unterschied von 1 ist in einer Berechnung schon sehr entscheidend. Bei einem Objekt der Charakteristik 1 wird immerhin 1 ganzes Objekt zu viel oder zu wenig gezählt. Es gibt eine Erweiterung des Integralbegriffes auf reelwertige Integralwerte, dies wird hier aber nicht weiter vertieft.

2.5.2 Konvergenz - Skalierung und Bildgenauigkeit

Um die Auswirkungen der diskreten Darstellung der Objekte vergleichen zu können, müssen die Sensornetzwerke das gleiche Sensoranzahlverhältnis zur Bildgröße und die gleiche Sensorverteilungsart ausweisen. Um dies bei verschiedenen Bildgrößen zu realisieren müssen die Eingabewerte skaliert werden. Wie in dem Programmteil in Punkt 8 unter Berechnungen 1 zu Bild 1 und 2 zu sehen ist, hat eine parallele Skalierung der Bildgröße und der Netzwerkskalierung wenig Auswirkung, da die gleichen Knotenwerte geliefert werden, und somit auch das gleiche Ergebnis. Die genauere Darstellung soll aber genutzt werden können. So können die Skalierungsfaktoren festgehalten werden, und nur die Bildgröße variiert werden. Somit werden Netzwerke mit sehr unterschiedlicher Anzahl an Sensoren verglichen.

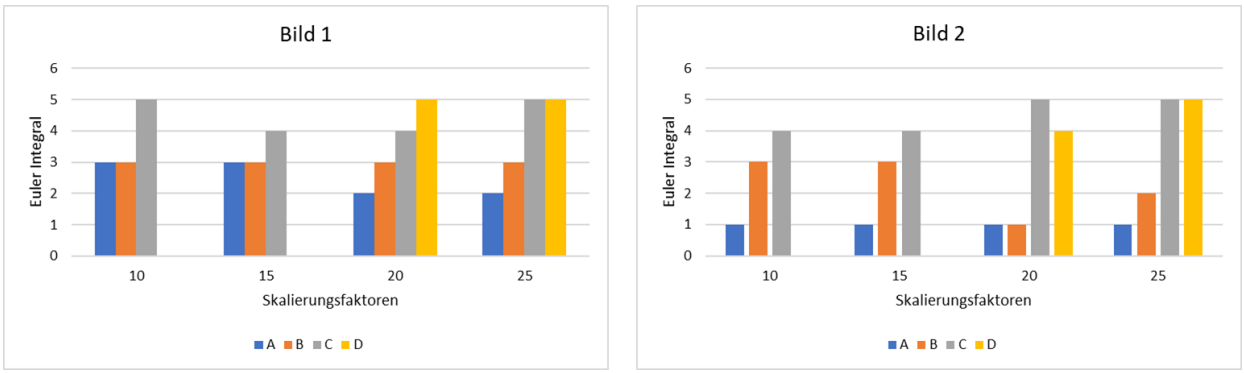

Abbildung 11: Vergleich der Integrale verschiedener Sensornetzwerke auf den Bildern 1 und 2. Die Sensornetzwerke sind verschieden im Verhältnis zur Bildgröße skaliert.

In der Abbildung 11 sind die Ergebnisse des Eulerintegrals zu den ersten zwei Bilder zu sehen. Dabei wurde die Bildgröße, und insbesondere parallel die Bildgenauigkeit variiert. Bildgröße A ist 50x50, B 100x100, C 500x500, und D 1000x1000. Der Skalierungsfaktor wurde dabei jeweils fest gelassen. Bild 1 hat fünf Rechtecke der Euler Charakteristik 1, so sollte das Integral 5 ergeben. Bild 2 hat vier Kreise der Euler Charakteristik 1, so sollte das Integral 4 ergeben. Somit nähern sich die Werte der Approximationen dem korrekten Wert fünf bzw. vier von unten aus an. Die Approximation gegen den richtigen Wert ist kein Zufall. Die Minimalen Charakterisiergraphen der Netzwerke mit dem richtigen Ergebnis stimmen überein:

Bild 1: (Gitter: Lokal, Bildgröße 500x500, Skalierungsfaktor: 10)

```
B1_GL_S10_C = berechnung_Minimaler_Charakterisierbaum(Bild_1_C,Gitter_linear_S
calierend(Bild_1_C,10))
B1_GL_S10_C
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

(Gitter: Lokal, Bildgröße 1000x1000, Skalierungsfaktor: 20)

```
B1_GL_S20_D = berechnung_Minimaler_Charakterisierbaum(Bild_1_D,Gitter_linear_S
calierend(Bild_1_D,20))
B1_GL_S20_D
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

(Gitter: Lokal, Bildgröße 500x500, Skalierungsfaktor: 25)

```
B1_GL_S25_C = berechnung_Minimaler_Charakterisierbaum(Bild_1_C,Gitter_linear_S
calierend(Bild_1_C,25))
B1_GL_S25_C
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

(Gitter: Lokal, Bildgröße 1000x1000, Skalierungsfaktor: 25)

```
B1_GL_S25_D = berechnung_Minimaler_Charakterisierbaum(Bild_1_D,Gitter_linear_S
calierend(Bild_1_D,25))
B1_GL_S25_D
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

Folgend werden diese vier Minimalen Charakterisiergraphen verglichen. Sie stimmen alle überein:

```
Vergleiche_Minimales_Gitter(B1_GL_S20_D[0],B1_GL_S20_D[1],B1_GL_S10_C[0],B1_GL
_S10_C[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

```
Vergleiche_Minimales_Gitter(B1_GL_S20_D[0],B1_GL_S20_D[1],B1_GL_S25_D[0],B1_GL
_S25_D[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

```
Vergleiche_Minimales_Gitter(B1_GL_S25_C[0],B1_GL_S25_C[1],B1_GL_S25_D[0],B1_GL
_S25_D[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

Bild 2: (Gitter: Lokal, Bildgröße 500x500, Skalierungsfaktor: 10)

```
B2_GL_S10_C = berechnung_Minimaler_Charakterisierbaum(Bild_2_C,Gitter_linear_S
calierend(Bild_2_C,10))
B2_GL_S10_C
```

```
([[1, 1, 1, 0, 1],
  [1, 1, 0, 0, 0],
  [1, 0, 1, 1, 0],
  [0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1]],
 [0, 1, 1, 2, 1])
```

(Gitter: Lokal, Bildgröße 500x500, Skalierungsfaktor: 15)

```
B2_GL_S15_C = berechnung_Minimaler_Charakterisierbaum(Bild_2_C,Gitter_linear_S
calierend(Bild_2_C,15))
B2_GL_S15_C
```

```
([[1, 1, 1, 0, 1],
  [1, 1, 0, 0, 0],
  [1, 0, 1, 1, 0],
  [0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1]],
 [0, 1, 1, 2, 1])
```

(Gitter: Lokal, Bildgröße 1000x1000, Skalierungsfaktor: 20)

```
B2_GL_S20_D = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_S
calierend(Bild_2_D,20))
B2_GL_S20_D
```

```
([[1, 1, 1, 0, 1],
  [1, 1, 0, 0, 0],
  [1, 0, 1, 1, 0],
  [0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1]],
 [0, 1, 1, 2, 1])
```

Vergleiche die drei aufgeführten Minimalen Charakterisiergraphen:

```
Vergleiche_Minimales_Gitter(B2_GL_S10_C[0],B2_GL_S10_C[1],B2_GL_S15_C[0],B2_GL
_S15_C[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

```
Vergleiche_Minimales_Gitter(B2_GL_S15_C[0],B2_GL_S15_C[1],B2_GL_S20_D[0],B2_GL
_S20_D[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

Dies Bedeutet zwar nicht direkt, dass die richtige Homologie erkannt wurde, es gibt jedoch eine hohe Wahrscheinlichkeit zusammen mit dem Fakt des richtigen Ergebnisses mehrfacher Sensornetzwerke mit unterschiedlicher Sensoranzahl.

Zu bedenken ist die stark wachsende Zahl an Sensoren, um Genauigkeit zu gewinnen, wie auch die gleichzeitig wachsende Genauigkeit der Bildapproximation. Dieses Problem führt auf folgende Vermutung:

Auf größeren Bildern können größere Netzwerke produziert werden. Dies führt zu der Verbesserung der Ergebnisse wie folgend zu sehen ist. Die geometrische Approximation bildet somit eine Art Grenze der Betrachtungsgenauigkeit. Die Vermutung, weniger die geometrische Schärfe, mehr die resultierende Genauigkeit des Sensornetzwerkes, dessen Größe in der

Umsetzung durch die Bildgröße begrenzt ist, beeinflusst die Genauigkeit des Ergebnisses, entsteht. Demnach reicht eine geometrische Bildschärfe, die alle Hohlräume des Objektes darzustellen vermag, aus. Eine entsprechende Verfeinerung des Netzwerkes bei gleichbleibender geometrischer Genauigkeit stellt folgende Grafik dar. Die Zahlen zeigen die Einträge einer 8x8 und einer 16x16 Bildmatrix der gleichen geometrischen Genauigkeit.

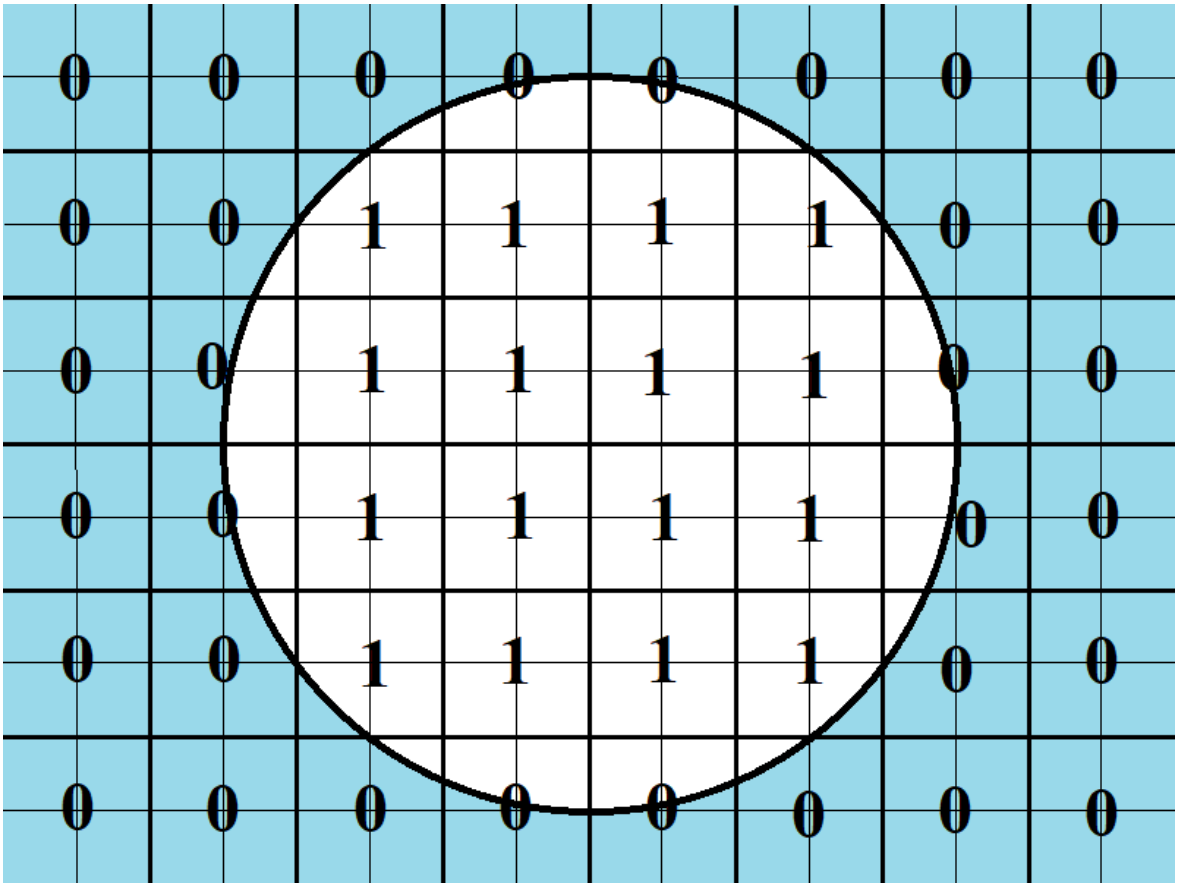

Abbildung 12: Grobe geometrische Darstellung der Höhenfunktion eines Kreises. Diese sehr grobe Approximation kann dann auch durch größere Matritzen dargestellt werden, ohne die geometrische Approximation weiter zu verfeinern.

Das Gitter erkennt also einen Kreis als Quadrat (Abbildung 12). Beide haben aber die gleiche Eulercharakteristik, somit reicht diese Darstellung im homologischen Sinne theoretisch aus. Soll ein größeres Gitter auf dieser geometrischen Approximation nutzen, kann man die Matrix wie in der Grafik zu sehen ist beliebig verfeinern, ohne die Verfeinerung für eine genauerer Darstellung zu nutzen.

Damit solch eine Betrachtung aber geschehen kann, gilt es erst die Auswirkungen verschiedener Netzwerkverbindungen zu ergründen.

2.5.3 Konvergenz - Verschiedene Gitter im Vergleich

In der folgenden Betrachtung werden verschiedene Verbindungsarten verglichen. Um eine Vergleichbarkeit zu gewährleisten werden die verschiedenen Verbindungen auf der gleichen Bildgröße und jeweils den gleichen Sensorwerten der gleichen Sensorpositionen. Zu beachten ist, dass der Skalierungsfaktor 3 ein sehr genaues Netzwerk liefert, der Faktor 15 ein sehr grobes. Es wird in er Abbildung 13 die Bildgröße [50x50] betrachtet. Die Exakten Werte sind im Programmteil unter 8.1 zu finden.

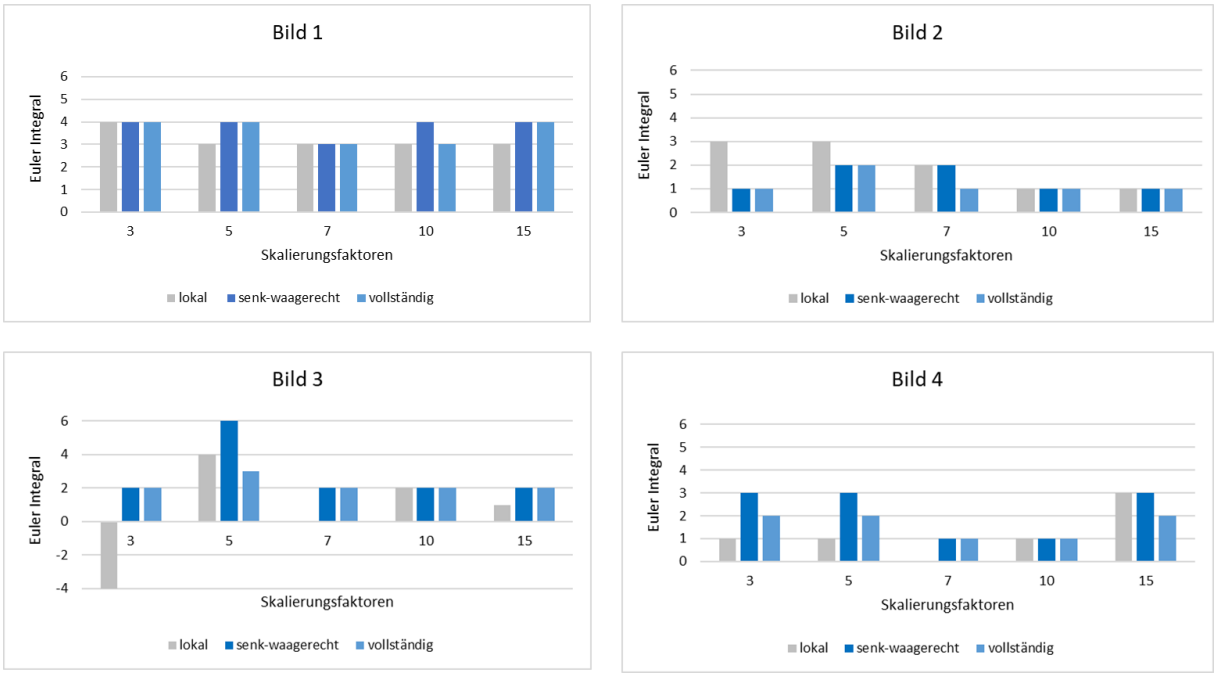

Abbildung 13: Berechnung des Euler Integrals auf den Bildern 1 bis 4 unter Verwendung verschiedener Netzwerkverbindungen und Skalierungsfaktoren. Die Netzwerke haben dabei je Skalierungsfaktor die gleichen Sensorpositionen.

Die geometrische Darstellung des Bildes ist nicht sehr genau, zudem kann kein Sensornetzwerk mit mehr als 50x50 Sensoren erstellt werden. Dennoch ist der Unterschied zwischen den Netzwerken groß. Teilweise unterscheidet sich der Wert um ein Vorzeichen. Keine der umgesetzten Verbindungen scheint eine eindeutige Konvergenz zu kreieren. Die Werte scheinen trotz großer Sprünge der Genauigkeit, wenn man allein die Anzahl der Sensoren betrachtet, die je Skalierungsfaktor verwendet werden, kaum der korrekten Lösung nahezukommen. Liegt dies an der Bildgröße, oder doch an der Genauigkeit? Wie verhalten sich spärliche Gitter auf sehr genauen Approximationen. Dazu wird ein Diagramm anhand der Bildgröße [1000x1000] erstellt (Abbildung 14). Die Skalierungsfaktoren sind entsprechend größer gewählt.

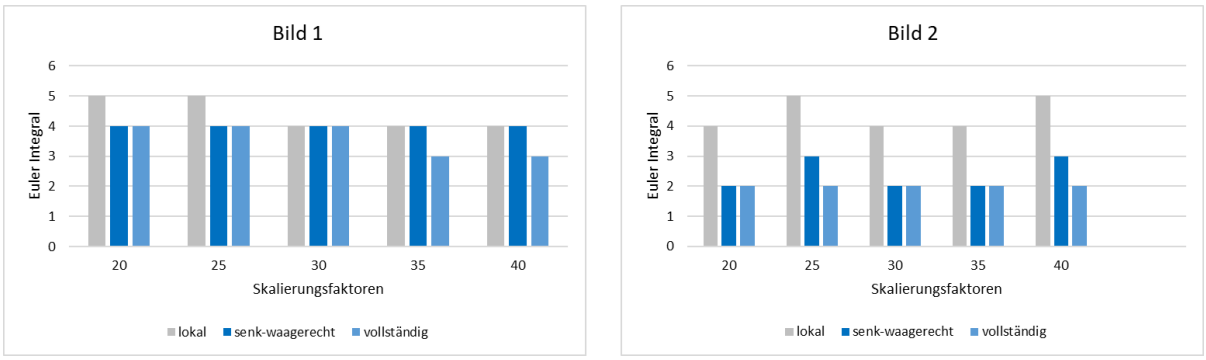

Abbildung 14: Berechnung des Euler Integrals auf den Bildern 1 und 2 unter Verwendung verschiedener Netzwerkverbindungen und Skalierungsfaktoren. Die Netzwerke haben dabei je Skalierungsfaktor die gleichen Sensorpositionen. Die Bildgröße ist diesmal statt 50x50 wie in Abbildung 13 1000x1000.

Für das erste Bild liefern die Netzwerke verschiedener Verbindungen annähernd konstante Werte. Bemerkenswert ist die Tatsache, dass einzig das lokale Netzwerk zumindest in den zwei Skalierungen mit den meisten Sensoren den korrekten Wert liefert, wenn dies auch nicht direkt eine korrekte Abtastung bedeutet. Um eine genauere Aussage über das Verhalten der Approximationen zu betrachten werden wieder die Minimalen Charakterisiergraphen berechnet. Zuerst wird Bild 1 betrachtet.

Der Minimale Charakterisierbaum des ersten Bildes auf einer 1000x1000 Matrix dargestellt durch das lokale Gitter mit Skalierungsfaktor 20.

```
Min_Git_B1_D_20 = berechnung_Minimaler_Charakterisierbaum(Bild_1_D,Gitter_linear_Scalierend(Bild_1_D,20))
Min_Git_B1_D_20
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

Das lokale Gitter mit Skalierungsfaktor 25 wird benutzt.

```
Min_Git_B1_D_25 = berechnung_Minimaler_Charakterisierbaum(Bild_1_D,Gitter_linear_Scalierend(Bild_1_D,25))
Min_Git_B1_D_25
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

```
Vergleiche_Minimales_Gitter(Min_Git_B1_D_20[0],Min_Git_B1_D_20[1],Min_Git_B1_D_25[0],Min_Git_B1_D_25[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

Vergleiche mit den Ergebnissen aus 2.5.2:

```
Vergleiche_Minimales_Gitter(B1_GL_S25_C[0],B1_GL_S25_C[1],Min_Git_B1_D_25[0],Min_Git_B1_D_25[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

Die lokal realisierten Verbindungen liefern also für die zwei feinsten Skalierungswerte die gleichen Minimale Charakterisiergraphen. Somit erfassen sie die gleich homologische Struktur. Insbesondere ergibt das daraus resultierende Integral den korrekten Wert. Zudem ergeben auch die Realisierungen aus 2.5.2 den gleichen Charakterisiergraphen. Wieder bedeutet dies keine absolute Sicherheit, die eigentliche Struktur richtig erkannt zu haben, ohne den eigentlichen Minimalen Charakterisiergraphen der Bilder zu kennen.

Unterschiede der Realisierungen sind hier im Verhältnis zu Bild zwei gering. Die lokale Version scheint gegenüber der waagerechten und senkrechten Realisierung und der vollständigen Realisierung einen leichten Vorteil zu haben. Deutlicher wird dies aber in Bild 2:

Der Minimale Charakterisierbaum des zweiten Bildes auf einer 1000x1000 Matrix dargestellt durch das lokale Gitter mit Skalierungsfaktoren 20, 25, 30, 35 und 40:

```
Min_Git_B2_D_20 = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Scalierend(Bild_2_D,20))
Min_Git_B2_D_20
```

```
([[1, 1, 1, 0, 1],
  [1, 1, 0, 0, 0],
  [1, 0, 1, 1, 0],
  [0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1]],
 [0, 1, 1, 2, 1])
```

Der Skalierungsfaktor 25 liefert hier kein korrektes Ergebnis. Es werden wie schon in 2.5.1 angedeutet mehr Zusammenhangskomponenten erkannt als in den Schritten vor und danach. Dies ist ein interessanter Sachverhalt, da die Struktur schon (durch 30 und 35) erkannt gewesen schien:

```
Min_Git_B2_D_25 = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Scalierend(Bild_2_D,25))
Min_Git_B2_D_25
```

```
([[1, 1, 1, 0, 1, 1],
  [1, 1, 0, 0, 0, 0],
  [1, 0, 1, 1, 1, 0],
  [0, 0, 1, 1, 0, 0],
  [1, 0, 1, 0, 1, 0],
  [1, 0, 0, 0, 0, 1]],
 [0, 1, 1, 2, 2, 1])
```

Nun wird wieder der gleiche Graph wie bei Skalierungsfaktor 20 geliefert.

```
Min_Git_B2_D_30 = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Scalierend(Bild_2_D,30))
Min_Git_B2_D_30
```

```
([[1, 1, 1, 0, 1],
  [1, 1, 0, 0, 0],
  [1, 0, 1, 1, 0],
  [0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1]],
 [0, 1, 1, 2, 1])
```

```
Min_Git_B2_D_35 = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Scalierend(Bild_2_D,35))
Min_Git_B2_D_35
```

```
([[1, 1, 1, 0, 1],
  [1, 1, 0, 0, 0],
  [1, 0, 1, 1, 0],
  [0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1]],
 [0, 1, 1, 2, 1])
```

Der Skalierungsfaktor 40 liefert ebenfalls kein korrektes Ergebnis.

```
Min_Git_B2_D_40 = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Scalierend(Bild_2_D,40))
Min_Git_B2_D_40
```

```
([[1, 1, 1, 0, 0, 1],
  [1, 1, 0, 0, 0, 0],
  [1, 0, 1, 1, 1, 0],
  [0, 0, 1, 1, 0, 0],
  [0, 0, 1, 0, 1, 0],
  [1, 0, 0, 0, 0, 1]],
 [0, 1, 1, 2, 2, 1])
```

Die lokalen Gitter auf der 1000x1000 Bildmatrix des zweiten Bildes mit Skalierungsfaktoren 20,30,35 sowie den Ergebnissen aus 2.5.2 liefert die gleichen Charakterisiergraphen, hier wird die Minimale Struktur aus 2.5.2 des zweiten Bildes auf einem 500x500 Bild mit Skalierungsfaktor 10 mit den Graphen der Lösung 5 auf dem 1000x1000 Bild verglichen:

```
Vergleiche_Minimales_Gitter(Min_Git_B2_D_20[0],Min_Git_B2_D_20[1],B2_GL_S10_C[0],B2_GL_S10_C[1])
```

```
'Die Minimalen Charakterisiergraphen stimmen überein!'
```

Bemerkenswerter Weise unterscheiden sich die Verschiedenen Gitter hier sehr deutlich. Die korrekte Lösung scheint nur von dem lokalen Gitter erkannt worden zu sein, wenn auch eine Unregelmäßigkeit beobachtet werden konnte. Die Art der Objekte scheint sich auf die Genauigkeit der Verschiedenen Gitter auszuwirken. Genauer betrachtet erkennen die nicht lokalen Verbindungen oft zu wenig Zusammenhangskomponenten, die lokale Version tendiert

auch mal dazu, zu viele zu erkennen. Genauer wird das durch die Betrachtung der Minimalen Charakterisiergraphen der nicht lokalen Verbindungen klar:

Hier wird Bild 2 mit senk und waagerechten Sensorverbindungen auf der 1000x1000 Matrix realisiert, der Skalierungsfaktor ist der feinste, der parallel betrachtet worden ist, hier 20 und zudem 25.

```
berechnung_approx_euler_ad(Bild_2_D,Gitter_linear_Schachbrett_Scalierend(Bild_2_D,20))
```

```
2
```

```
Min_Git_B2_D_20_S = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Schachbrett_Scalierend(Bild_2_D
Min_Git_B2_D_20_S
```

```
([[1, 1, 1], [1, 1, 1], [1, 1, 1]], [0, 1, 2])
```

```
berechnung_approx_euler_ad(Bild_2_D,Gitter_linear_Schachbrett_Scalierend(Bild_2_D,25))
```

```
3
```

```
Min_Git_B2_D_25_S = berechnung_Minimaler_Charakterisierbaum(Bild_2_D,Gitter_linear_Schachbrett_Scalierend(Bild_2_D
Min_Git_B2_D_25_S
```

```
([[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 0], [1, 1, 0, 1]], [0, 1, 2, 2])
```

Wie bereits erwähnt erkennet das Netzwerke der senkrechten und waagerechten Sensorverbindungen zu wenige Zusammenhangskomponenten. Trotz hoher Genauigkeit des Bildes ist der Wert des Integrales im Gegensatz zur lokalen Realisierung dem eigentlichen Ergebnis entschieden entfernter. Der betrachtete Vergleich ist dabei aber nicht in direkter Weise auf allgemeine Bilder übertragbar. Wie bereits angemerkt ändert sich der Verhalten des Integrals anhand vieler Faktoren, insbesondere durch die Komplexität der Höhenfunktion. Somit kann ein anderes Gebiet möglicherweise für die lokale Realisierung der Sensorverbindungen sehr schlechte Werte liefern. Zudem gilt es die Auswirkungen der geometrischen Approximation genauer zu untersuchen, wie in z.B. in 2.5.2 schon angedeutet wurde. Die Konvergenz des Verfahrens, wenn man denn bei dieser Anzahl an Proben überhaupt davon reden kann, scheint zudem nicht sehr linear zu verlaufen.

Damit ist nicht nur der ganzzahlige Wertebereich gemeint, sondern auch die Unregelmäßigkeiten der Werte bei Gitterverfeinerung. Der herkömmlichen Approximation nach wird folgendes erwartet: Je feiner das Gitter wird, desto näher an der Lösung liegt das Ergebnis. Zudem erwartet man oft eine gewisse Nähe der Approximationen ähnlicher Eingabewerte. Dies ist aber nicht klar zu erkennen. Obwohl die lokale Approximation auf dem 1000x1000 Bild 2 schon bei dem Skalierungsfaktor 35 das richtige Ergebnis und den gleichen Minimalen Charakterisierbaum wie die genauste Berechnung mit den Faktoren 20 oder auch bei 30 ergibt. Dennoch weicht die Berechnung des Faktors 25 wieder vom Ergebnis ab. Wird

der Minimalen Charakterisiergraphen genauer betrachtet, so erkennt dieser erstaunlicher Weise eine Komponente nicht als zusammenhängend. Sie wird mehrfach gezählt.

Dies könnte aber mit entsprechender Realisierung einer oder mehrerer zusätzlicher Verbindungen geschehen, indem so die Komponenten vernetzt werden. Keine neuen Sensorknoten werden theoretisch benötigt. Wie dabei allerdings auf die richtigen Verbindungen geschlossen werden kann ist unklar. Das Realisieren von allgemein mehr Verbindungen ist nach der oberen Betrachtung auch nicht unbedingt generell zielführend. Die Ergebnisse werden nicht genauer, sondern verschlechtert sich in dieser Betrachtung mehr, zumindest in den oberen Beispielen. Dies bietet eine Interessante Forschungsbasis. Gitter, dessen Sensoren in einer lokalen Umgebung in Form eines Kreisen verbunden sind, können eine interessante Betrachtung darstellen. Somit kann vielleicht dem Probleme aus 2.5.1 entgegengewirkt werden, ohne dass zu viele Sensorverbindungen entstehen. Dieses Problem resultiert aus fehlenden Verbindungen von Sensoren in bestimmte Richtungen, entgegen der realisierten Verbindungen.

Lokale Gitter bieten so einen weiteren interessanten Ansatz, der in Abschnitt 2.7 behandelt wird. Diese Idee ist dabei sehr lokal. Die Aussage über die lokale Güte der Approximation kann ein Sensor nur liefern, wenn er nur mit Sensoren verbunden ist, die in seiner geometrischen Nähe sind. Deshalb wird im nächsten Abschnitt das Verhalten des „Nachbardurchschnittes“ auf lokalen Gittern betrachtet. Möglicherweise ist es sinnvoll, zuerst die Idee in Abschnitt 2.7.1 zu lesen.

2.5.4 Lokale Gitter und Nachbarschaftsverhältnisse

Die Funktion Anzahl_gleichwertiger_und_verschiedenwertiger_Nachbarn liefert zu jedem Sensor die Anzahl der direktverbundenen Sensoren, die den gleichen Sensorwerte aufweisen, und die Anzahlen der Sensoren mit verschiedenwertigen und verschiedenartigen Sensorwerten. Betrachtet man nun die Summe dieser Anzahlen über alle Sensoren des lokalen Netzwerkes und teilt diese Summen jeweils durch die Anzahl an Sensoren des Netzwerke, so bildet das Verhältnis des Durchschnittes der gleichwertigen zu dem Durchschnitt der verschiedenwertigen Sensorwerten das Nachbarverhältnis. Es wird somit berechnet:

$$\left(\frac{Summe \text{ der Anzahlen gleichwertiger Nachbarn}}{(Anzahl\ Sensoren)}\right) / \left(\frac{Summe \text{ der Anzahlen verschiedenwertiger Nachbarn}}{(Anzahl\ Sensoren)}\right)$$

Ein Vorteil dieses Wertes ist, dass die Berechnung ohne die Berechnung des Euler Integrals oder des Charakterisiergraphen möglich ist. Die Berechnung ist auch wesentlich weniger aufwendig. Der Annahme aus 2.7.1 nach sollte der Wert größer werden, je besser das Verhältnis

der Gittergröße zur Gebietskomplexität passt und je angepasster die Wahl der Gitterpunkte dem Bild gegenüber ist.

Um diesen Verhalt auf den Grund zu gehen, werden zu Bild eins und zwei in den Bildgrößen A =[50x50], B=[100x100], C=[500x500], D= [1000x1000] und den Skalierungsfaktoren 3, 5, 7, 10, 15, 20 für Bild 1 und 10, 15, 20 für Bild 2 der entsprechende Wert berechnet. Die Abbildung 15 zeigt dabei die Auswirkungen der Netzwerkskalierung und gleichzeitig der Bildskalierung.

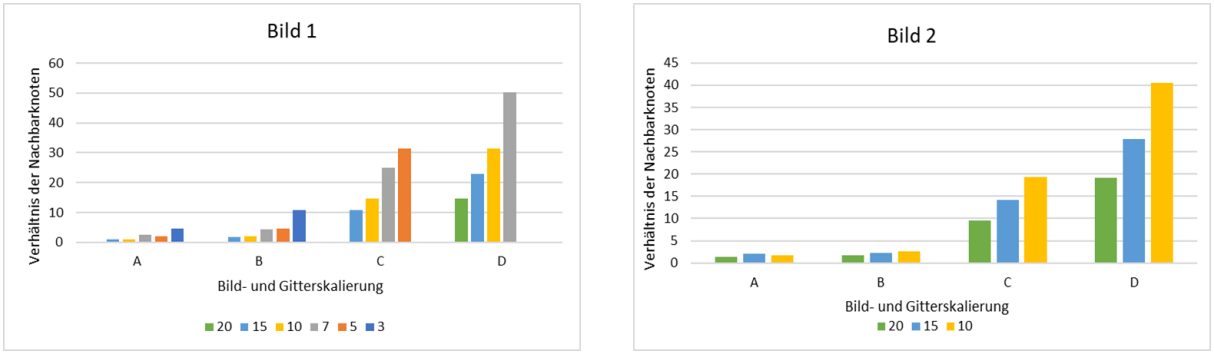

Abbildung 15: Das Berechnete Verhältnis der Nachbarsensoren der Bilder 1 und 2 in verschiedenen Bildgrößen. Dabei ist nicht immer ein Wert angegeben, da die Rechenleistung teilweise nicht ausreichte. Verschiedene Skalierungen der Bilder in verschiedenen Größen werden der Bildgröße und dem Skalierungsfaktor nach sortiert.

Hier werden, sobald die Berechnung nicht die Rechenleistung überschritten hat, die zugehörigen Werte des Integrals aufgeführt (Abbildung 16):

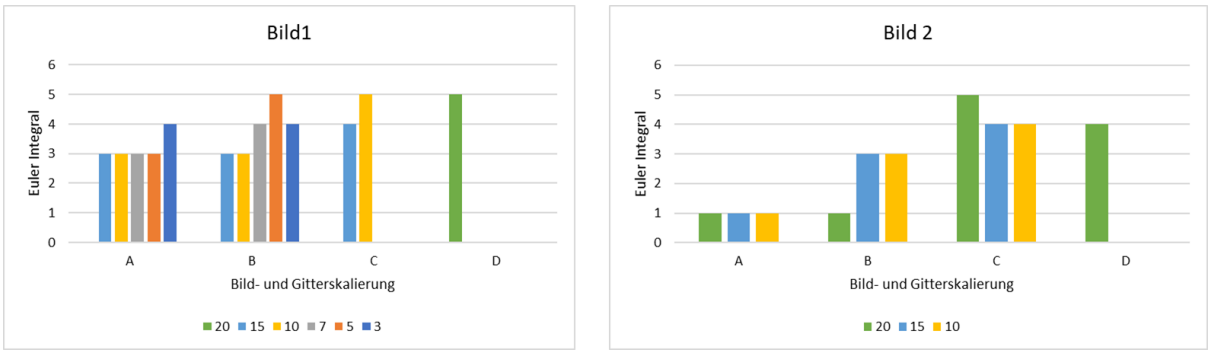

Abbildung 16: Die zu Abbildung 15 gehörigen Integralwerte in gleicher Anordnung. Auch hier sind nur werte angegeben, dessen Berechnung im Ramen der Reschen Leistung lag.

Die Vermutung scheint sich in Abbildung 15 zu bestätigen, wird aber Abbildung 16 hinzugezogen, fallen einige Unregelmäßigkeiten auf. Der Zusammenhang richtiges Ergebnis zu Nachbarverhältnis scheint nicht direkt zu existieren. Zwar nähern sich die Werte vor allem bei der Bildskalierung dem korrekten Ergebnis an, jedoch gibt es Fälle nahezu gleichen

Verhältnisses mit unterschiedlichen Werten, Gittern und Bildgrößen. Da der Vermutung nach nicht nur die Güte der Approximation, sondern auch das Verhältnis der Sensorzahl zur Bildkomplexität ausdrückt, scheint dies aber noch kein großes Problem darzustellen. Teilweise liefern die gleichen Integralwerte sogar den gleichen Verhältniswert wie in Bild 1, Grundlage B, Faktor 3 und Bild 1, Grundlage C und Faktor 15. Ein größeres Problem stellen Verhalte wie in Bild 1 auf Grundlage B. Der Wert scheint für Faktor 3 signifikant höher als für den Skalierungsfaktor 5, obwohl 5 das richtige Ergebnis liefert. Die Abweichung um den Wert 1 ist zwar bei ganzzahliger Betrachtung nicht groß, dennoch ist der Wert des Verhältnissen entsprechend um einiges größer. Liegt dies nun daran, dass eine Falsche Struktur erkannt wurde, die das korrekte Ergebnis zufällig geliefert hat? Um diesen Sachverhalt genauer zu betrachten werden die Minimalen Charakterisiergraphen dieser Netzwerk Bild Kombinationen berechnet: (Faktor: 3)

```
ung_Minimaler_Charakterisierbaum(Bild_1_B,Gitter_linear_Scalierend(Bild_1_B,3))
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2])
```

(Faktor: 5)

```
ung_Minimaler_Charakterisierbaum(Bild_1_B,Gitter_linear_Scalierend(Bild_1_B,5))
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

Verglichen mit dem Charakterisiergraphen Bild 1, Bildgrundlage D, Skalierungsfaktor 20:

```
ng_Minimaler_Charakterisierbaum(Bild_1_D,Gitter_linear_Scalierend(Bild_1_D,20))
```

```
([[1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
  [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
  [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0],
  [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]],
 [0, 1, 2, 1, 3, 2, 4, 3, 1, 2, 1, 2, 2, 1])
```

Dies zeigt, dass für den Faktor 3, obwohl viel mehr Sensoren genutzt werden, Zusammenhangskomponenten zusammengezogen wurden, und ein anderer Graph wie in Skalierungen mit weit weniger Sensoren entsteht, sowie das Ergebnis abweicht. Ob diese nun richtigerweise zusammengezogen wurden, und sich noch eine andere Ungenauigkeit dahinter verbürgt, oder ob die Zusammenhangskomponenten nicht zusammengehören, ist dabei unklar. Der erste genannte Fall würde den höheren Wert des Verhältnisses erklären, und auch den abweichenden Integralwert rechtfertigen. Der Vermutung nach hat das Netzwerk mit Faktor 3 somit eine Ungenauigkeit mehr erkannt, und somit würde ein hoher Wert möglicherweise gerechtfertigt sein. Auch Fehler, wie von Krupa (Krupa, 2012) beschrieben, können durch die Art der Zielträger auftreten. Da Bild 1 aber Vierecke realisiert, die die Matrix vollständig darstellen kann, und insbesondere keine Kanten verwendet, mit denen die beschriebenen Fehler auftreten können, ist in diesem Fall nicht davon auszugehen. Würden die Kanten nicht senk oder waagerecht der Bildmatrix verlaufen, so können auch für Rechtecke diese Verhalte auftauchen. Die eigentliche Fehlerursache ist aber sehr schwer zu beurteilen, da vor allem Berechnungen für nicht sehr viele Werte durchgeführt werden können, da die Rechendauer und Rechenleistung für große Bilder sowie Netzwerke enorm ist. Oft hilft es noch, erst den Charakterisiergraphen aufzurufen, und dann erst das Integral zu berechnen. Dies ist keine Zeitersparnis, aber somit lässt sich unnötig tiefe Rekursion vermeiden. Zusammengefasst ist eine großangelegte Kalkulation zur genaueren Erkenntnisgewinnung hilfreich.

Auf diese Art zum Beispiel kann somit die erarbeitete Testumgebung, die der Programmteil darstellt, genutzt werden, um diese Art von Erkenntnissen zu erlangen. Dazu gilt aber zu klären,

welche Probleme diese Umsetzung mit sich bringt. Dies wird im folgenden Abschnitt behandelt.

2.6 Probleme der Untersuchung

2.6.1 Approximative Grundlage

Wie oben bereits angedeutet, ist ein 'Bild' in diesem Falle die Darstellung der resultierenden Höhenlinien einer Menge von 2-dimensionalen Objekten auf endlich vielen quadratischen 'Pixeln'. Somit ist ein 'Bild' bereits eine Abbildung, deshalb auch die Namenswahl 'Bild', also eine Approximation des eigentlich darzustellenden Problems. Lässt man nun die Menge der Objekte in einem 'Bild' berechnen, und stimmt sie nicht mit der im Bild eigentlich hinterlegten Anzahl an Objekten überein, so bedeutet dies noch nicht, dass ein Fehler vorliegt. Dies ist für die Überprüfung der Korrektheit sowohl für die weitere Untersuchung der Approximationsfähigkeit ein Problem. Des Weiteren ist die Untersuchung nur Zielführend, wenn der Code auch genau die Umsetzung der mathematischen Methoden realisiert.

2.6.2 Lösungsstrategien

Wahl von geeigneten Größenunterschieden:

1. *Bildgröße:* Wählt man ein sehr großes Bild, also mit sehr vielen 'Pixeln', und darauf sehr große (einfache) Objekte, so stellt das Bild die Objekte geometrisch sehr genau dar. Natürlich kann das Problem so nicht vollständig beseitigt werden, aber lassen sich zumindest Schlüsse ziehen, ob es entweder gut konvergiert oder sehr schlecht bzw. nicht konvergiert. Zwischen sehr schlecht und überhaupt nicht lassen sich in diesem Falle dann aber wieder keine Rückschlüsse treffen.

2. *Gitterweite:* Der Unterschied der Gitterweite zur Bildgröße kann variiert werden, obwohl das Problem gleichbleibt. Das 'Bild' kann geometrisch linear genauer, also schärfer machen, also durch mehr Pixel darstellen lassen. Das gleiche Problem auf genaueren Daten mit skaliertem Gitter kann somit überprüft werden. Das Testen der Approximation des Integrales kann somit ausgereizt werden, ohne mit der Gitterapproximation in den Bereich der Bildapproximation zu kommen. Realisiert wird dies durch lineare Skalierung des Bildes mit den Objekten und dem Gitter.

Die Eigenschaft der Konvergenz:

Vermutet man eine gute Approximierbarkeit, so folgt eine gute Konvergenz der Approximation. Findet man bei der Untersuchung mehrerer genauer werdender Approximationen des gleichen 'Bildes' keinen Zusammenhang, und stellt dies bei

verschiedenen 'Bildern' fest, obwohl man wie oben beschrieben zum Beispiel auch die Größenverhältnisse skaliert hat, so ist zumindest davon auszugehen, dass das Verfahren sehr schlechte oder keine Konvergenzeigenschaften besitzt. Verschiedene Bilder meinen dabei insbesondere anderer Pixelanzahl, Genauigkeit, Objekten mit verschiedenen Euler Charakteristiken und verschiedener Anzahl von Objekten, die mal alle getrennt, mal überlappend, mal komplett ineinander verschachtelt sind. Somit kann eine genauere Untersuchung des Integrales realisiert über *Theorem 4.3* auf diese Weise zum Erfolg führen.

2.7 Geeignete Sensorpositionen

2.7.1 Idee

Ein diskretes approximierendes Sensornetzwerk mit lokalen Verbindungen enthält, wenn auch die Güte der Approximation nicht bekannt ist, Informationen über die Eigenschaften des approximierten Gebietes, also auch Informationen darüber, wie gut die Approximation lokal wohl ist. Gemeint ist damit folgendes: Einem diskretes Sensornetzwerk, welches ein Gebiet zu approximieren versucht, indem es gleichmäßig verteile Sensoren nutzt, enthält Informationen über die lokale Güte der Approximation. Ein Sensor, dessen direkte Nachbarn ausschließlich den gleichen Sensorwert haben, approximiert mit hoher Wahrscheinlichkeit seine Umgebung ganz gut. Ein Sensor, dessen direkte Nachbarn sehr unterschiedliche Sensorwerte haben, und der (wenige) keine gleichwertigen direkten Nachbarn hat, approximiert seine Umgebung wahrscheinlich nicht sehr genau. Dabei ist vorauszusetzen, dass genügend Sensoren im Verhältnis zur Komplexität des Gebietes zur Verfügung stehen, und das Netzwerk dementsprechend auch kein Minimales anstrebt, da die Informationen der „überflüssigen“ Sensoren gebraucht werden. Deshalb ist die Annäherung an eine exakte Minimale Struktur nicht direkt umsetzbar. Erst müssen kritische Sensoren Umgebungen verfeinert werden, um die ungenaueren Umgebungen besser abzutasten. Bei Sensoren mit „vielen“ gleichen Nachbarn kann hingegen auf eine Verfeinerung verzichtet werden. Ein solches Gitter sollte dabei somit möglichst weit von einem Minimalen entfernt sein, bis die genaue Struktur (genau genug) erfasst werden konnte, also kein neuer Sensor das darauf berechnete Minimale Netzwerk mehr verändert. Dann kann bei Bedarf das Gitter minimiert werden, um ein exakten Minimale Charakterisiergraphen zu erhalten. Die folgende Funktion gibt die Anzahl gleichwertiger und verschiedenwertiger Nachbarknoten sowie unterschiedlicher verschiedener Nachbarsensoren wieder. Somit kann ein approximierendes Sensornetzwerk auf Grund dieser Daten bewertet werden. Zudem kann die beschriebene Form der Vorgehensweise darüber realisiert werden.

2.7.2 Umsetzung (in Python 3)

Funktion: Anzahl_gleichwertiger_und_verschiedenwertiger_Nachbarknoten

In:

```
  Knotenwerte; Liste mit Knotenwerten,
               Knoten i hat den Knotenwert: (Knotenwerte[i])
  Adjazenz Liste; 2-dimensionale Liste, Knoten i kennt Knoten j, falls
      i in Adjazenz Liste[j] und j in Adjazenz Liste[i]
```

Out:

```
  Liste; anzahl_gleichwertiger_Nachbarn, Liste mit Einträgen entsprechend der Anzahl der dire
kt
  benachbarten Knoten mit gleichem Knotenwert
  Liste; anzahl_verschiedenwertiger_Nachbarn, Liste mit Einträgen entsprechend der Anzahl der
direkt
  benachbarten Knoten, die nicht den gleichen Knotenwert besitzen
  Liste; anzahl_verschiedenartiger_nachtbarn, Liste mit Einträgen entsprechend der Anzahl der
direkt
  benachbarten Knoten mit unterschiedlichen Knotenwerten, also der Anzahl, wie viel verschied
enwertige
  Knoten er kennt
```

Kommentar:

z1: Variable, Knoten i hat anzahl_gleichwertiger_Nachbarn[i] gleichwertige Nachbarn.

z2: Variable, Knoten i hat anzahl_verschiedenwertiger_Nachbarn[i] Nachbarn mit anderen Knotenwerten.

z3: Variable, Knoten i hat anzahl_verschiedenartiger_nachtbarn[i] verschiedenwertige Nachbarknoten (s.o.)

z4: Für jeden Knoten j berechne:

```
  z5: Setze den Listeneintrag von Knoten i der anzahl_gleichwertiger_Nachbarn Liste auf -1, d
a
      der Knoten sich selbst kennt, dies aber nicht mitgezählt werden soll.
  z6/7: Setze die Listeneinträge von anzahl_verschiedenwertiger_Nachbarn und
        anzahl_verschiedenartiger_nachtbarn auf 0.
  z8: Temporäre Variable zum Speichern der verschiedenen Knotenwerte der direkt benachbarten
Knoten
  z9: Für jeden Knoten j den Knoten i kennt:

      z10/11: Falls Knoten i und j gleiche Knotenwerte haben, erhöhe anzahl_gleichwertiger_Na
chbarn[i]
              um 1
      z12: Falls nicht:

          z13: erhöhe anzahl_verschiedenwertiger_Nachbarn[i] um 1
          z14-16: Falls der Knotenwert noch nicht vorgekommen ist, erhöhe
          anzahl_verschiedenartiger_nachtbarn[i] um 1 und merke den Knotenwert.
```

z17: Gebe anzahl_gleichwertiger_Nachbarn, anzahl_verschiedenwertiger_Nachbarn, anzahl_verschiedenartiger_nachtbarn zurück.

```
def Anzahl_gleichwertiger_und_verschiedenwertiger_Nachbarknoten(knotenwerte, adjazenzliste):
    anzahl_gleichwertiger_Nachbarn = []
    anzahl_verschiedenwertiger_Nachbarn = []
    anzahl_verschiedenartiger_nachtbarn = []
    for i in range(0,len(knotenwerte)):
        anzahl_gleichwertiger_Nachbarn = anzahl_gleichwertiger_Nachbarn + [-1]
        anzahl_verschiedenwertiger_Nachbarn = anzahl_verschiedenwertiger_Nachbarn + [0]
        anzahl_verschiedenartiger_nachtbarn = anzahl_verschiedenartiger_nachtbarn + [0]
        temp = []
        for j in adjazenzliste[i]:
            if(knotenwerte[j] == knotenwerte[i]):
                anzahl_gleichwertiger_Nachbarn[i] = anzahl_gleichwertiger_Nachbarn [i] + 1
            else:
                anzahl_verschiedenwertiger_Nachbarn[i] = anzahl_verschiedenwertiger_Nachbarn[i] + 1
                if(knotenwerte[j] not in temp):
                    temp = temp + [knotenwerte[j]]
                    anzahl_verschiedenartiger_nachtbarn[i] = anzahl_verschiedenartiger_nachtbarn[i] + 1
    return anzahl_gleichwertiger_Nachbarn, anzahl_verschiedenwertiger_Nachbarn, anzahl_verschiedenartiger_nachtbarn
```

3. Zusammenfassender Ausblick

3.1 Zusammenfassung und Ausblick

In dieser Arbeit wird eine nummerische Testumgebung geschaffen, die eine genauere Betrachtung des Euler Integrals für Zählprobleme in der zweidimensionalen Ebene ermöglicht. Dabei sind Probleme wie die Darstellung kontinuierlicher Zielträger mittels diskreter Höhenfunktion sowie ungünstiger Sensorverbindungen aufgetreten. Verschiedene Zielträger und verschiedene Sensorverbindungen beeinflussen die Konvergenz des Integrales. Zudem wurde die Minimierung der Netzwerkgraphen als Untersuchungsfunktion nummerisch umgesetzt und eine Vermutung für lokale Netzwerkverbindungen eingeführt, welche Aussage über die Netzwerkeigenschaften bezüglich der Höhenfunktionen der Zielträger liefern soll. Dabei sind verschiedene Arten von Zielträgern betrachtet worden, darunter Rechtecke, Kreise, Ringe und Rechtecke mit rechteckigem Loch. Diese wurden auf Bildmatritzen verschiedener Größen abgebildet, und durch Netzwerke verschiedener Verbindungen ausgewertet. Darauf kann die Eigenschaft der Auswertungen anhand der Integralwertes und des Minimalen Charakterisiergraphen betrachtet werden. Verschiedene Arten von Fehlerquellen können auftreten, zudem ist das Integral ganzzahlig.

Dies motiviert weitere Untersuchungen durchzuführen, zum Beispiel das Verhalten von geometrischen diskreten Zielträgerdarstellungen durch Netzwerke verschiedener Sensoranzahl, ohne die geometrische Darstellung gleichzeitig zu verfeinern. Des Weiteren können verschiedene Arten von Bildgrößen, Sensornetzwerken, Verbindungstypen und Sensorpositionen getestet werden. Zudem kann die Idee der lokalen Verfeinerung eines Netzwerkes, welches geometrisch lokale Verbindungen realisiert, zur Verbesserung der diskreten Approximation und das Verhältnis der direkten Sensorverbindungen im Netzwerk mit gleich und verschiedenwertigen Sensorwerten getestet werden.

4. Referenzen

Baryshnikov, Yury, & Ghrist, R. (2009). Target enumeration via euler characteristic integrals. SIAM Journal on Applied Mathematics, 70(3), 825–844. https://doi.org/10.1137/070687293

Baryshnikov, Yury, & Ghrist, R. (2010). Euler integration over definable functions. Proceedings of the National Academy of Sciences of the United States of America, 107(21), 9525–9530. https://doi.org/10.1073/pnas.0910927107

Betti, E. (1870). Sopra gli spazi di un numero qualunque di dimensioni. *Annali di Matematica* 4, 140–158 https://doi.org/10.1007/BF02420029

Hatcher, A. (2002). Algebraic Topology, *Cambridge University Press*, Cambridge,

Heap, B. R. (1963). Permutations by Interchanges, *The Computer Journal*, Volume 6, Issue 3, Pages 293–298, https://doi.org/10.1093/comjnl/6.3.293

Kneser, (1926). Die Topologie der Mannigfaltigkeiten, Jahresbericht DMV, Band 34, S. 1–14

Krupa, S. (2012). Numerical Analysis of Target Enumeration via Euler Characteristic Integrals: 2 Dimensional Disk Supports. http://arxiv.org/abs/1202.2934

Tanaka, K. (2017). Minimal networks for sensor counting problem using discrete Euler calculus. Japan Journal of Industrial and Applied Mathematics, 34(1), 229–242. https://doi.org/10.1007/s13160-017-0243-2

5. Plagiatserklärung

Plagiatserklärung der / des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über

Anwendung des Euler Integrals auf Zählprobleme

selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

_____

(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

_____

(Datum, Unterschrift)